Five Tales of Random Forest Regression

by

Samuel Carliles

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

October, 2016

Abstract

We present a set of variations on the theme of Random Forest regression: two applications to the problem of estimating galactic distances based on photometry which produce results comparable to or better than all other current approaches to the problem, an extension of the methodology to produce error distribution variance estimates for individual regression estimates which property appears unique among non-parametric regression estimators, an exponential asymptotic improvement in algorithmic training speed over the current de facto standard implementation which improvement was derived from a theoretical model of the training process combined with competent software engineering, a massively parallel implementation of the regression algorithm for a GPGPU cluster integrated with a distributed database management system resulting in a fast roundtrip ingest-analyze-archive procedure on a system with total power consumption under 1kW, and a novel theoretical comparison of the methodology with that of kernel regression relating the Random Forest bootstrap sample size to the kernel regression bandwidth parameter, resulting in a novel extension of the Random Forest methodology which offers lower mean-squared error

ABSTRACT

than the standard methodology.

Primary Reader: Alex Szalay

Secondary Reader: Carey Priebe

Tertiary Reader: Tamás Budavári

Acknowledgments

My deepest and most heartfelt gratitude goes to the Sloan Foundation, the Gordon and Betty Moore Foundation, and the NSF.

Thanks to Bruce Sommerfield; Kerry Wayne, Jr.; Darwin Johnson; Kelly Parker; Mark Branson; David Campbell; Jason Amato; Henry Hoffman; Ron DePiro; John Barbuto; Andrew Gross; Philip Jones; Ronald Emery; Tawny Holm; Seth Sanders; Chris Rollston; Ellen Robbins; P. Kyle McCarter; Jerrold Cooper; K Rudolph; Antoinette Marie Burton; Theo Schlossnagle; Dwight Wilson; Joanne Selinski; Jonathan Eisenhamer; Amy Germida; Sunil Soprey; William Hanke; Maureen McSharry; Clinton Adams; Susan Weiss; Manuel Barrueco; Michael Koehrsen; Nikita Kuzmin; Linda Rorke; Donniell Fishkind; Christian Skalka; Scott Smith; Gerald Masson; Cathy Thornton; Debbie DeFord; Zack Burwell; Steve Rifkin; Steve DeBlasio; John Shep-

pard; Steven Zucker; C.Y. Chien; Rao Kosaraju; Suzanne Jutila; Ross Gutmeier; Nabil Abdo; Marcel Poisot; Eric Goldlust; Omar Ahmad; Sanjeev Khudanpur; Noah Smith; Markus Dreyer; David Smith; Roy Tromble; Misha Kazhdan; Noah Cowan; Ann Forster; Joel Schildbach; the Holstein clan; Rick Lewis and the staff at Academy Animal Hospital; Judith Caparossi and the staff at the Maryland SPCA; Sadie Lingham; Margie Gier; Lenalee Fulton; Tara Hentgen; Jan Vandenberg; Alainna White; Josh Greenberg; Jordan Raddick; Nolan Li; Jayant Gupchup; Manuchehr Taghizadeh-Popp; Michael Specian; Tamas Szalay; László Dobos; George Fekete; Richard Wilton; Ching-Wa Yip; Sebastien Heinis; István Csabai; Tamás Budavári; Ani Thakar; Tor Thilert; Steve Williams; Sundar Nathikudi; David Graff; Zack Dixon; Justin Musterman; Ryan Woodard; Catherine Williams; and Sam Seljan.

I must single out for particular thanks Man-Nyon Kim; Samuel Carliles, Jr.; Chad Kersey; Alex Szalay; and Carey Priebe. Lastly, I must thank Joan Miller for bearing with me for so long.

All of these people have suffered my ignorance, arrogance, presumption, dismissiveness, impatience, and impetuousness with patience and grace. I hope that, at the very least, they got a few good laughs watching me progress so slowly. Cheers!

ACKNOWLEDGMENTS

Dedication

Contents

List of Tables

List of Figures

Chapter 1

Greetings and Felicitations

The present work began as an inquiry into what benefits might come from applying what was, at the time, a relatively new nonparametric regression technique – Random Forest regression – to the problem of estimating galactic distances using data from the Sloan Digital Sky Survey [66]. For a subset of the galaxies present in the Sloan survey, we have distances measured accurately using spectroscopy. These distances are measured in redshift, effectively the magnitude of the doppler shift between light emitted from the galaxy and the light that we observe, having shifted at a known rate in distance terms. There is known to be a functional relationship between the broadband magnitudes of light emitted from galaxies and their distance from earth observers. Thus we may combine our spectroscopic redshifts with broadband magnitudes to apply supervised machine learning techniques to the problem of learning this functional relationship. Initially we sought to do so accurately, and to

extend the methodology slightly to yield confidence intervals on our estimates. This allows us to provide distances to all galaxies in the Sloan survey, which in turn allows others to produce a reasonably accurate map of the density of galaxies in the universe around us out to a redshift of about 0.2, the practical limit of the Sloan telescope's range. This initial foray into photometric redshift estimation using Random Forests is chronicled in chapter 2. Shortly thereafter, we found Random Forest regression useful in demonstrating that a galaxy's orientation with respect to us – the extent to which we view it edge-on versus on its face – is a useful feature in improving the accuracy of such photometric redshift estimates. This is described in chapter 3.

After all this, we began to feel discontent with the limitations of the available Random Forest implementations. At the time there were really only the original FORTRAN code and the implementations available in R and Weka. We had been using the R implementation, which was groaning under the volume of data we had; it made seemingly poor use of system memory, so that it struggled even with volumes of data which should have been manageable on a given system, and as we show in chapter 4, the slope of the training time grew precipitously as we added each new observation. At the same time, we had been grappling with the limitations of the contemporary standard scientific computing architecture: keep data in a giant conventional RDBMS, retrieve interesting subsets of the data small enough to be

analyzed on a desktop workstation, analyze, plot, then optionally write results back to the RDBMS. Picking at workstation-size subsets began to feel overly constraining, and yet, even simply to *move* the data to a system sufficient to analyze it was infeasible over existing network architecture. We had long since concluded that the issue was the absurd notion that we should store data in one place and analyze it in another; no, on further reflection it made more sense to begin analyzing the data *in situ*. Since we had been using Microsoft SQL Server, and since Microsoft had added the ability to write custom extensions to SQL Server's functionality using an integrated Common Language Runtime (SQL CLR), we resolved to write our own Random Forest regression implementation in C# .NET with the intention of integrating it with SQL Server. The resulting work is described in chapter 4.

By the time this C# implementation was minimally functional, we had begun hitting a power wall with our science cluster efforts. To wit, our research group had already undertaken significant retrofitting of our building's power and cooling infrastructure! This was interpreted as a sign that we needed to take the thermodynamic constraints on scientific computation more seriously. News that Google planned to open a data center near the Arctic[1] strengthened the case. Facebook has since done likewise[2]. We had not the resources to build such a data center, so instead we at-

[1] https://www.google.com/about/datacenters/inside/locations/hamina/

[2] https://www.facebook.com/notes/lule%C3%A5-data-center/lule%C3%A5-goes-live/474321655969861/

tempted design of a new type of computation cluster, based on low power nodes with weak CPUs and many-core GPUs supporting general purpose computation. This required a different approach to implementation which we describe in chapter 5.

Finally we returned to theoretical considerations of the technique, proposing and demonstrating a relation to kernel regression with consequences for parameter tuning and ensemble construction which allowed us to reduce the regression error on a synthetic data set, and which should apply just as well to many real world data sets. These results are described in chapter 6.

Enough pleasantries; let the mayhem begin!

Chapter 2

Random Forests for Photometric Redshifts

2.1 Introduction

The redshifts of extragalactic sources are accurately determined from spectroscopic measurements. Spectroscopy, however, is limited by the high wavelength resolution, which can only partly be overcome with more observing time. Recently, an increasing number of studies rely on less precise statistical estimates of the redshifts based on more efficient broadband photometry. In fact, these *photometric redshifts* are in the core of many key projects of the upcoming survey telescopes. Various succesful techniques have been developed. Some leverage training sets [e.g., 12, 18, 19, 60], others utilize template spectra for comparisons [e.g., 6, 7, 13, 17, 32, 34, 41, 52] and

some use a combination of the two [15, 23]. Applied to the same data sets these techniques by and large converge, reaching similar accuracy, primarily limited by the systemtic errors in the data. Most redshift estimators today fall short in providing reliable models of the uncertainties. All things being equal, a technique that offers a verifiable model of the estimation error is preferable to one without. Some work on error estimators which analyze performance post hoc has been done, e.g., by Oyaizu et al. [47]. In this paper, we focus on a new empirical technique borrowed from the arsenal of the machine learning community; a method that has intriguing statistical properties that makes it well-suited for photometric redshift estimation.

The structure of the paper is as follows. In Section 2.2, our choice of method called Random Forest regression is introduced for the problem of redshift estimation. In Section 2.3, we apply this new technique to a well-studied data set from the Sloan Digital Sky Survey [SDSS; 66] and discuss the results. Section 2.4 concludes our study.

2.2 Random Forests

Empirical redshift estimation can be viewed as a regression problem, if one believes that the redshift is a function of the photometric observables, e.g., the apparent magnitudes in various passbands. Several parametric and non-parametric methods have been applied to this problem but most are geared toward accuracy and loose

control of the uncertainties. Our approach is to focus on the error properties of the estimates, which we achieve by using a method called Random Forest regression [hereafter RF; 10]. The idea is to build independent regression trees on the training set, and to utilize the resulting distribution for characterizing the error and deriving an accurate estimate for each object.

2.2.1 Regression trees

Regression trees [11] are used for modeling continuous functions. The trees are built by a deterministic procedure that recursively partitions the training set into a hierarchy of clusters of similar objects. This hierarchy is represented (and stored in an implementation) as a binary tree, whose nodes contain the collections of sources. New nodes of the tree are created by splitting the nodes and their collections in an optimal way. The split at each node is done along one of the axes of the input space (e.g., the SDSS *ugriz* magnitudes), and the choice of which dimension is best to split on is done according to which dimension gives the lowest *resubstitution error* in the resulting subsets. The resubstitution error is equivalent to the standard deviation from the mean along the direction of the desired parameter, i.e., the known spectroscopic redshift $z_{\rm spec}$, summed over the two new subsets,

$$\epsilon_{\rm resubs} = \epsilon_{\rm left} + \epsilon_{\rm right} \tag{2.1}$$

with components

$$\epsilon_{\text{left}} = \frac{1}{N_{\text{left}}} \sum_{z_i \in \text{left}} (z_i - \bar{z}_{\text{left}})^2 \tag{2.2}$$

$$\epsilon_{\text{right}} = \frac{1}{N_{\text{right}}} \sum_{z_i \in \text{right}} (z_i - \bar{z}_{\text{right}})^2 \tag{2.3}$$

where N_{left} and N_{right} are the numbers of objects on the two sides of the splitting point, and $\bar{z}_{\text{left}}$ and $\bar{z}_{\text{right}}$ are the means of those respective collections. Along each dimension there is an optimal split point which will minimize the above score in equation (2.1). We choose the best axis according to resubstitution error and we split the node accordingly. The reason for computing the resubstitution error around the mean is that the mean is the optimal parameter estimator for the response (in our case the redshift) of the objects in a given cluster; that is, if you had to pick a scalar value to represent the redshift of all objects in the cluster, the mean would be the optimal choice according to a Euclidean distance metric. We choose to minimize the resubstitution error for the same reason; the resulting splits are optimal according to Euclidean distance. One could try a more robust estimator than the mean, but the mean works well in practice, it's easy to compute, and the behavior of regression trees constructed thusly is well understood. There is also a nice intuitive clustering analogue: choosing a split point in this way can be seen as simply doing k-means clustering with $k = 2$.

Regression trees are typically grown fully, that is, until each leaf node contains

only one value, and then *pruned* back to optimize performance on cross-validation data sets. If branching at a given node does not improve performance on test data, branches from that node are cut off.

2.2.2 Randomized trees

Random Forests are ensembles of regression trees trained on *bootstrap* samples. Given a training set D of size N, a bootstrap sample is a subset of D selected by choosing N objects from D with replacement [31]. The idea is that one can generate various bootstrap samples and train separate predictors (in our case, build regression trees) with those samples to produce many different estimates. One can then average these estimates into a more robust aggregate. This is called *bootstrap aggregating*, or *bagging* [9], and in addition to giving an accurate estimate, it also provides a distribution of the individual estimates. It has been shown that bagging predictors using bootstrap samples drawn from the population reduce variance [36, p. 247]. The assumption then is that this holds to some extent with bootstrap samples drawn from the data as is the only option in the real world. In practice the amount of variance reduction gained per additional tree is determined empirically for each dataset by increasing the forest size until the variance on a heldout test set appears to converge to a lower limit.

Random Forest regression trees also introduce some additional randomization beyond what results from the bootstrap process. At each node, rather than splitting

on the best dimension from the input space, the split is done on the best dimension from a random subspace [38]. For example, out of the u, g, r, i, and z dimensions, a node might randomly choose only three of these dimensions to consider, say u, r, and z. Each node chooses its random subspace independently of all other nodes, including parent nodes. This random subspace method helps to increase independence between trees, and it has the additional benefit of reducing computational cost. Each tree branch is grown until a user-specified minimum number of training objects (commonly five) is reached in a node, and the tree does not branch any further from that node. Its value is then defined as the mean of the known redshifts associated with the objects it contains. After all trees in a RF are grown, the forest can predict responses to new input points. A query point is classified left or right starting at the root of each tree in the forest, moving to the next level until it reaches a leaf node, and the aggregate estimate $z_{\rm phot}$ for the new object is defined as the mean of these leaf node values.

2.3 Application to SDSS galaxies

Now we turn to apply the above method to multicolor observations of galaxies in the SDSS. The SDSS is two surveys in one: it is a photometric survey that takes multi-color images of the sky on the best nights in five passbands u, g, r, i, z, and it is also a spectroscopic survey that spends most of the time measuring the spectra

of close to a million objects. For our exercise, we use a subset of the Main Galaxy Sample [MGS; 55] in the Data Release 6 catalog [3].

2.3.1 Sample selection

To ensure high data quality we use a strict set of selection criteria. The ranges are chosen to eliminate erroneous measurements that are obvious outliers. We expect Random Forests to be somewhat robust to missing and erroneous data as the randomization process reduces the reliance on any particular data element. Breiman and Cutler describe several possible approaches to dealing with missing values[1]. However, our primary concern is the development of good estimates with error distributions, so we choose to take advantage of the large amount of complete and accurate data available in SDSS. We perform the final selection of the training and test sets using the SDSS Science Archive stored in a SQL Server database engine. The Catalog Archive Server is searched via the online CasJobs site[2] using the following SQL command.

```
SELECT a.SpecObjID, a.ObjID, a.PrimTarget, a.z, ...
FROM SpecPhoto a
   JOIN UberCal b ON b.ObjID = a.ObjID
WHERE a.SpecClass = 2 -- Galaxies
   AND a.SciencePrimary = 1
   AND a.ZConf > 0.9 AND a.ZErr < 0.1
   AND a.z BETWEEN 0.0001 AND 1
   AND (a.ZWarning & 0xFFFF1B10) = 0
   AND (a.PrimTarget & 448) > 0
   AND a.ModelMag_U > -9999 AND ...
```

[1] http://www.stat.berkeley.edu/∼breiman/RandomForests/cc_home.htm
[2] http://casjobs.sdss.org/CasJobs/

```
    AND b.ModelMag_U > -9999 AND ...
GO
```

Here we select highly confident redshifts, extinction corrected magnitudes including ones using the so-called *über-calibration* [50] and the sets of flag bits from `PrimTarget`, which allows us to select the appropriate target categories. In particular, we select only galaxies, chosen both from the SDSS Main Galaxy Survey (MGS) and the Luminous Red Galaxies (LRGs).

2.3.2 Results

We constructed a forest of 400 trees trained on $80,000$ objects to estimate redshifts for $100,000$ previously held-out test objects. For a given test point, a forest with B trees provides B estimates of the redshift, $\{z_i\}$. Then for this particular input, the aggregate estimate for the redshift, z_{phot}, is the mean of these z_i estimates. We also evaluated the trimmed mean (eliminating those z_i outside of 2σ of their mean,) and the results were virtually identical. The RMS error between trimmed means and corresponding spectroscopic redshifts is 0.023. The character of our estimates over the usable range for our methodology is shown in Figure 2.1a. The estimates are generally good, with some slight bias visible near the origin due to the local skewness of the underlying distribution of redshifts. The average difference between z_{phot} and z_{spec} is shown as a function of z_{phot} in Figure 2.1b. This shows that over the usable range, given what we believe z_{phot} should be, we are just as likely to err low or high,

meaning that what $z_{\rm spec}$-dependent bias we have is dominated by variance in the same $z_{\rm spec}$ neighborhood.

For a given test point, each z_i estimate has an associated estimation error, $\epsilon_i \equiv z_i - z_{\rm spec}$, and we define the aggregate estimation error as

$$\epsilon_{\rm phot} \equiv z_{\rm phot} - z_{\rm spec}, \tag{2.4}$$

which is equivalent to the mean of the ϵ_i values. We can think of the z_i as realizations of identically distributed random variables with some physical mean. Since $z_{\rm phot}$ is the sample mean of these z_i, there is a central limiting behavior; $z_{\rm phot}$ tends toward the physical mean. Under ideal conditions ($B \to \infty$, independence among the z_i) the Central Limit Theorem would give us the distribution from which $z_{\rm phot}$ is drawn. If the mapping from color to redshift space were non-degenerate, the physical mean would be equivalent to $z_{\rm spec}$, and this would give us the distribution from which $\epsilon_{\rm phot}$ is drawn, i.e., the distribution of our estimation error. Following this intuition and applying it to our SDSS galaxy sample leads us to a useful estimate of this distribution. To wit, we observe that

$$\frac{\epsilon_{\rm phot}}{\sigma_\epsilon} \sim N(0,1) \text{ approximately} \tag{2.5}$$

where σ_ϵ indicates the standard deviation of the tree error realizations ϵ_i. Figure 2.2a shows a histogram of errors standardized and plotted along with the Gaussian distri-

bution. The agreement is striking, though there is a slight skew which is anticipated by Figure 2.1a; the galaxy distribution is more concentrated at lower values of z_{spec}, where we tend to overestimate. Still, the agreement is remarkable, even if not perfect. As an additional sanity check, we can compute the percentage of our standardized observed errors that fall within the level-α critical values for a given α to see how well this compares with $1 - \alpha$, the area under the standard normal curve between those critical values. For instance, if $\alpha = 0.05$, the area under the lower tail of the standard normal curve is $\alpha/2 = 0.025$ as is the area under the upper tail, thus the area between the tails is 0.95. We therefore expect 95% of our standardized errors to fall between the boundaries delimiting the tails under the assumption that our errors are normal. We do this test for several values of α and plot the results in a quantile-quantile plot in Figure 2.2b. Again, though the results are not perfect, they are very close to what we expect. Figure 2.2a anticipates that not quite as many of our standardized errors are near zero as should be, so for instance roughly 61% of our errors fall within the critical values for $\alpha = 0.32$, corresponding approximately to the range within one standard deviation of zero, where we expect to see 68%. Notwithstanding this imperfection, Figure 2.2b indicates that our errors fall within prescribed bounds with nearly the correct probability.

We wish to emphasize that though we standardize our errors for the purposes of analyzing their behavior in the aggregate, before standardization they are intrinsically unique *per-object* error distribution estimates. The value of σ_ϵ is unique to each new

test object; it reflects something about the quality of the input data and our confidence in our estimate for this particular observation.

2.3.3 Practical details of the procedure

In separate tests from those described above, we measure how Random Forest performance scales with data and with forest size. We generate training sets of size $80,000$; $40,000$; $20,000$; $10,000$; and $5,000$ by first uniformly sampling the full result set without replacement, and then uniformly sampling the resulting subset (again without replacement), so that smaller training sets are proper subsets of each larger set.

For each training set size, we train eight forests of one hundred trees each. Since final estimates can be aggregated from any number of tree predictions, this allows us to observe how the quality of estimates scales with forest size; in our case, from 100 to 800 trees. It also has the additional benefit of allowing the computation to be done on a machine with a "modest" amount of memory. Constructing trees with different training set sizes, of course, allows us to observe how the quality of estimates scales with training set size.

We observe the accuracy one could expect to see for various training set and forest sizes. We trained RFs of 100 trees on training sets of size $5,000$; $10,000$; $20,000$; $40,000$; and $80,000$ using colors from the über-calibration. Performance is then tested on a random subset of $10,000$ held-out objects. The resulting RMS

error over these 10,000 objects is shown in Figure 2.3a. The training times for these forests are shown in Figure 2.3b. The gain in accuracy bought by larger training sets is significant. Predictably, so is the gain in training time. Since on average one can train one tree as fast as another with the same amount of data, one does not need a plot to see that training time scales linearly with the number of trees. In our tests all work was done on an Apple MacBook equipped with a 2.0GHz Intel Core 2 Duo processor with 2GB RAM using R version 2.5.1 with the Random Forest package version 4.5-18[3].

Next we tested the effect of increasing the number of trees in the forest. We trained eight Random Forests of 100 trees each on a training set of 80,000 objects using ubercal colors. For our random subset of 10,000 held-out objects, we computed aggregate estimates using individual predictions from first one tree, then two trees, then three, and so on for effective forest sizes of 1 up to 800. The resulting RMS error over these 10,000 objects is shown in Figure 2.3c. For this training set with the bootstrap sample size we used, the RMS error on this test set stops improving significantly beyond the first 50 trees. With each additional tree the forest converges toward a limiting error which is intrinsic to the data. Between 50 and 800 trees the gain in RMS error is only about 1%, and certainly by the time we had trained 200 trees we had reached a point of diminishing returns. One should note that the rate of

[3]The R statistical computing environment, as well as the Random Forest package for R, may be downloaded from http://www.r-project.org/. Sample R code along with a small subset of our DR6 data selection suitable for demonstrating the methodology is available at http://www.sdss.jhu.edu/∼carliles/photoZ/.

convergence will depend on the forest parameters such as bootstrap sample size and training sample size.

2.4 Discussion

The performance of Random Forests on the SDSS MGS is comparable to that of other machine learning methods, e.g., artificial neural networks [48]. Measured over the entire test set, the Random Forest error is nearly mean zero, i.e. Random Forests would appear to give nearly zero bias. However, as is clearly visible in Figure 2.1a, there is boundary bias - a clear tendency to overestimate near the low end of the z range and a slightly less obvious tendency to underestimate near the high end. Correcting this bias is a subject for future study. It should also improve the performance of our error estimates as in Figure 2.2. We suspect, however, that Random Forests and other empirical methods may have come close to a lower bound on error achievable by treating photo-z as a regression problem, described in more detail in [14]. Extending the Random Forest technique to provide a redshift distribution estimate rather than a single scalar estimate will likely be the subject of future work.

Our error estimation method appears to perform comparably to the NNE method of [47]. Both methods yield normally distributed errors with accurate per-object parameter estimates, allowing arbitrary choice of α values for confidence intervals. Though the direct association of our error estimates with their corresponding test

objects is obvious (the error distribution parameters are computed explicitly in the same process using the same data as the redshift estimate), in an indirect way, the NNE yields a similar association - the error is assumed to be similar to the error of nearby objects in magnitude space. But we prefer our method for several reasons.

First, there is something inelegant about estimating the error in a step separate from the redshift estimation. Indeed it is not clear why one method would be preferred for redshift estimation and another for error estimation; they are the same problem - one is just the linear translation of the other. If one can measure the true distance of one's redshift estimate from the spectroscopic value, then one can just as easily apply that same methodology to estimate the redshift directly. Deferring to an error estimator like the NNE estimator described by [47] is a tacit admission that any redshift estimate is of limited use without the confidence given by an empirical error estimator. But if one believes one's error estimator, and that error estimator shows that the error is non-zero as was the case in several of the trials published by [47], then one must conclude that the redshift estimator is biased. The only reasonable thing to do next would be to subtract that error from the initial redshift estimate and use the result as the final estimate instead. Random Forests do this intrinsically, yielding zero-mean error estimates and error distribution variance estimates which apparently fall out of the process for free. A theoretical explanation for why this process works is forthcoming, and the explanation turns out to be quite subtle.

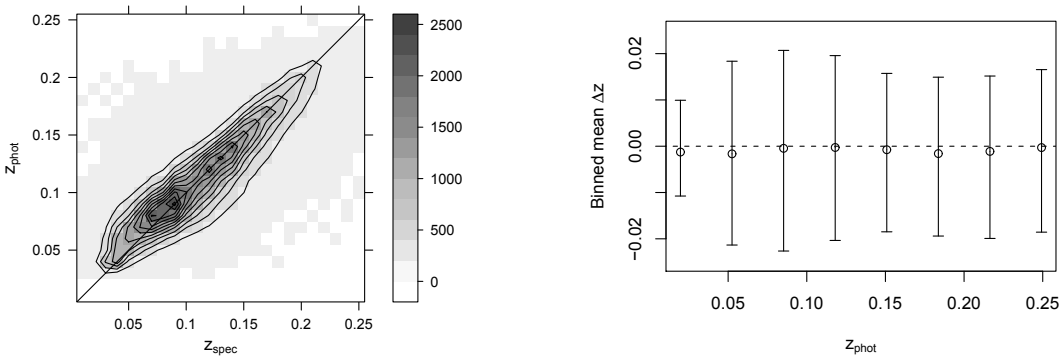
Our main reason to prefer Random Forests is the existence of proofs of convergence

and bounds on error [10]. Yet another advantage is that in the course of developing our own forthcoming Random Forest code, we have discovered that Random Forests can be implemented in a computationally efficient way, and that consequently, redshifts estimated using Random Forests and their associated error estimates can be computed with very low computational overhead.

It should be noted that as an empirical method, Random Forests cannot extrapolate beyond the limits of the training data. That is, for instance, methodologies which utilize a geometric model like template-fitting can exploit assumptions about the underlying geometry of the problem, and a geometric model fit to training data can be used to extrapolate beyond the observations. Measuring the quality of these extrapolations is difficult or impossible. One could not apply any empirical method (for instance, NNE) beyond the range of the observations and still expect reliable behavior. But these extrapolated estimates could be scientifically useful nonetheless in the absence of anything better.

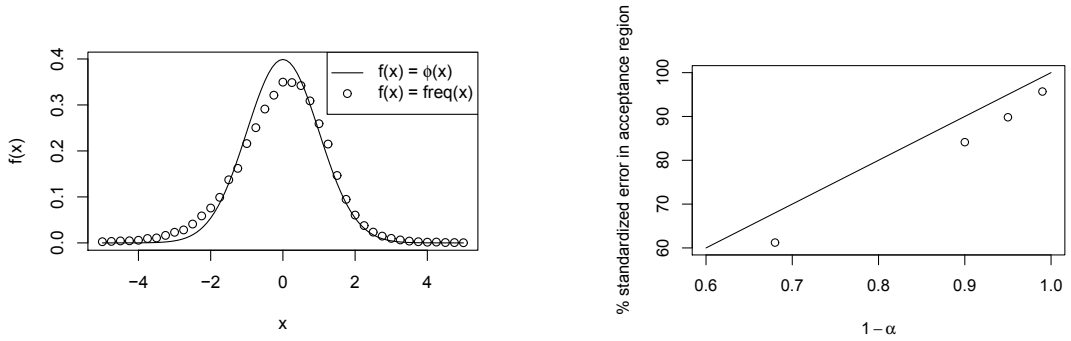
Random Forest regression improves the utility of redshift estimates by giving us good measurements of the estimation error, and thus compares favorably to other methods giving comparable estimates where extrapolation is not expected. Care should be exercised in estimating the per-object variance of the estimation error, and it may be necessary to estimate a scaling factor of the variance estimates empirically for each new training set and Random Forest configuration. Extrapolation is a special case of the more general problem of non-representativeness of training data, and this

will likely be a target of future study. It will also be useful to try to improve the quality of the estimates. One possible approach to this will be to weight training object contributions according to quality measures that accompany the training data (for instance, the magnitude error columns in the case of SDSS). Given the performance and the reliable per-object estimation error distributions offered by Random Forests, they represent an attractive alternative to other photo-z methodologies.



(a) (b)

Figure 2.1 (a) Photometric vs. spectroscopic redshift for 100,000 test objects distributed into 25 bins along each axis with 7 levels. (b) Mean error for 100,000 test objects in eight bins vs. photometric redshift with bars marking region containing 34% of errors on either size of the mean.



(a) (b)

Figure 2.2 (a) Observed standardized error ($\epsilon_{\text{phot}}/\sigma_\epsilon$) for 100,000 test objects binned (circles) and Standard Normal distribution (curve). (b) Percent observed standardized error within level-α critical values for 100,000 test objects vs. $1-\alpha$ (circles) and percent error expected within level-α critical values vs. $1-\alpha$ (line).

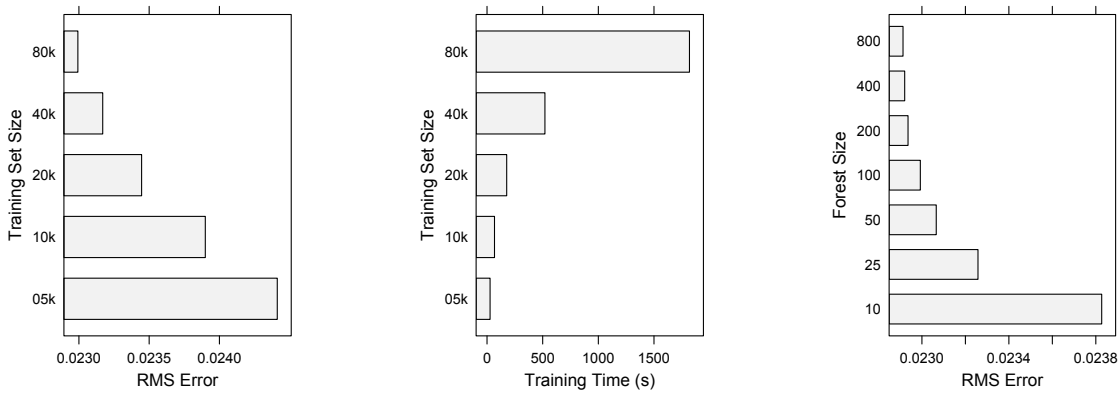


Figure 2.3 (a) RMS error for estimates on $10,000$ test objects given by forests of 100 trees trained on $5,000$; $10,000$; $20,000$; $40,000$; and $80,000$ objects. (b) Training time for a forest of 100 trees trained on $5,000$; $10,000$; $20,000$; $40,000$; and $80,000$ objects. (c) RMS error for estimates on $10,000$ test objects given by forests of 10 to 800 trees trained on $80,000$ objects.

Chapter 3

Effect of Inclination of Galaxies on Photometric Redshift

The inclination of galaxies induces both reddening and extinction to their observed spectral energy distribution, which in turn impact the derived properties of the galaxies[1]. Here we report a significant dependence of the error in photometric redshift (photo-z) on the inclination of disk galaxies from the Sloan Digital Sky Survey. The bias in the photo-z based on the template-fitting approach increases from

[1]We thank Andrew Connolly, David Koo, Istvan Csabai, Samuel Schmidt, Rosemary Wyse, and Brice Ménard for comments and discussions. We thank the referee for helpful comments and suggestions. We acknowledge support through grants from the W.M. Keck Foundation and the Gordon and Betty Moore Foundation, to establish a program of data-intensive science at the Johns Hopkins University.

This research has made use of data obtained from or software provided by the US National Virtual Observatory, which is sponsored by the National Science Foundation.

Funding for the SDSS and SDSS-II has been provided by the Alfred P. Sloan Foundation, the Participating Institutions, the National Science Foundation, the U.S. Department of Energy, the National Aeronautics and Space Administration, the Japanese Monbukagakusho, the Max Planck Society, and the Higher Education Funding Council for England. The SDSS Web Site is http://www.sdss.org/.

-0.015 in face-on to 0.021 in edge-on galaxies. A Principal Component Analysis on the full sample of photometry reveals the inclination of the galaxies to be represented by the 2nd mode. The corresponding eigenspectrum resembles an extinction curve. The isolation of the inclination effect in a low-order mode demonstrates the significant reddening induced on the observed colors, leading to the over-estimated photo-z in galaxies of high inclinations. We present approaches to correct the photo-z and the other properties of the disk galaxies against the inclination effect.

3.1 Motivation

The inclination of galaxies has been used as a tool to infer the opacity of disk galaxies [e.g., 8, 26, 27, 29, 33, 39, 40, 59]. The effect of the inclination on disk galaxies are twofold: the reddening and the extinction on its spectral energy distribution, supported by many of the recent studies based on large samples of galaxies [e.g., 5, 21, 30, 44, 49, 53, 58, 63]. If these effects are not corrected for, one would expect an impact on the derived properties of the galaxies. One such property is the photometric redshift (photo-z) of a galaxy, because it relies on the observed colors and magnitudes [e.g., 19, 41] of the galaxy.

Many panoramic sky surveys will measure primarily broadband photometry of galaxies. Considering how the distance to a galaxy bares its influence from the inferred properties of the galaxy to the large scale structures in the universe, the correct

estimation of the photo-z of galaxies is of utmost importance. Studies in cosmology are also impacted by the accuracy in the redshift of galaxies of various inclinations. Notably, Marinoni and Buzzi [45] have recently constrained dark energy content with statistics of the inclination of galaxies in pairs where the redshifts are known. We therefore explore and quantify in this work the dependence of the error in the photo-z on the inclination of disk galaxies. Among all of the Hubble morphological types, the geometry of disk galaxies deviates substantially from the spherical symmetry. One would expect a relatively large amplitude in any inclination-dependent effect.

We present the sample of disk galaxies in §3.2. We quantify the photo-z error as a function of the inclination of the galaxies in §3.3. We present approaches to correct the photo-z and the other properties of the disk galaxies against the inclination effect in §3.4.

3.2 Sample

The galaxies in this study constitute a volume-limited sample from the Sloan Digital Sky Survey [SDSS; 66] in which the redshift ranges from 0.065 to 0.075 and the r-band Petrosian absolute magnitude ranges from -19.5 to -22. To construct this sample we use the same selection criteria as described in Yip et al. [63, hereafter Paper I], in which the authors derived the extinction curves of star-forming disk galaxies from the SDSS spectroscopy in the Data Release 5 [DR5, 2]. In this work

we consider instead the DR6 [3], because of the improved photometric calibration and the larger number of galaxies. The other main characteristic of the sample is that, in the above ranges of redshift and absolute magnitude, the distribution of the inclination of the disk galaxies is uniform (see Figure 2b of Paper I). As such, the properties of the galaxies are not biased from one inclination to the next. There are 6285 galaxies in total in the analysis, a 36% increase from DR5.

We follow Paper I and use the r-band apparent minor to major axis ratio (b/a, or the derived parameter "expAB_r" in the SDSS) as a proxy for the inclination of the galaxies. The uncertainty in using b/a as an inclination measure is considered by simulating 2D sky projections of the disk galaxies, where each galaxy is modeled as a triaxial spheroid at various *known* inclinations. With the premise that the disk galaxies are nearly circular [supported most recently by 51, who obtained average face-on ellipticity of 0.16 for a sample of disk galaxies in the SDSS] and negligible disk scale height, not surprisingly we come to a similar conclusion as Shao et al. [53] that the apparent axis ratio is a good measure for the inclination of disk galaxies. This conclusion is drawn based on the positive correlation of the simulated inclination and the apparent axis ratio. We decide to discuss the details of the simulation in a separate paper due to the limited space here.

We consider inclination ranges 0.0–0.2[2], 0.2–0.3, 0.3–0.4, 0.4–0.5, 0.5–0.6, 0.6–0.7, 0.7–0.8, 0.8–0.9, and 0.9–1.0 when calculating the photo-z statistics. To follow

[2]Here we are effectively considering galaxies with inclinations from 0.1–0.2 because there are only 2 galaxies with inclinations from 0.0–0.1. We however determine to set the inclination range of the first bin to be 0.0–0.2, so that all of the edge-on galaxies can be included.

the convention in the SDSS all of the spectral energy distributions are expressed in vacuum wavelengths.

3.3 Dependence of Photometric Redshift Error on Inclination of Disk Galaxies

3.3.1 Photo-z Error vs. Inclination

The photo-z error, z(photo)−z(spec), as a function of the inclination of the disk galaxies is shown in Figure 3.1. Three cases are considered: in Figure 3.1(a) the SDSS photo-z based on the template-fitting[3] approach [24, and references therein]; in Figure 3.1(b) the SDSS photo-z by using the Artificial Neural Network approach [48], in which the authors used an implementation similarly to that of Collister and Lahav [18]; and in Figure 3.1(c) the photo-z calculated in this work based on the Random Forest approach [16, details are deferred to §3.4.3]. The "CC1" photo-z of Oyaizu et al. [48] are used, because they were obtained by employing only 4 SDSS colors $u-g$, $g-r$, $r-i$, and $i-z$ in the training procedure, in this sense similar to Figure 3.1(c). While both the bias ($\langle \mathrm{z(photo)} - \mathrm{z(spec)} \rangle = -0.004 \pm 0.001$, 0.003 ± 0.001, 0.003 ± 0.0003) and the root mean square (RMS $= \sqrt{\langle [\mathrm{z(photo)} - \mathrm{z(spec)}]^2 \rangle}$

[3]The galaxies with photo-z bias < -0.05 in Figure 3.1(a) may be a result of larger uncertainty in the colors. The one-sigma uncertainty in $u - g$ for those with bias < -0.05 is 0.12 ± 0.19, and that for bias > -0.05 is smaller by about half, 0.065 ± 0.058.

$= 0.032 \pm 0.002,\ 0.020 \pm 0.001,\ 0.018 \pm 0.0005$)[4] are respectively of the same order of magnitudes for all of the cases, the dependences on the inclination are noticeably different. In the template-fitting approach the bias in the photo-z increases from the face-on to edge-on galaxies, in such a way bias(edge-on)$-$bias(face-on) $= 0.036$. The photo-z bias also changes sign with inclination, showing that it is the ensemble bias ($= -0.004 \pm 0.001$) being minimized instead of the bias for a particular group of galaxies. The statistics of the photo-z error in all of the inclination bins are given in Table 3.1. In contrast, the photo-z error does not show prominent dependence with the inclination of the disk galaxies in both of the machine learning approaches (Figure 3.1(b) and 3.1(c)). Because the inclination is not included explicitly in the training procedure in both of these approaches, this lack of inclination dependence is interpreted as the success of the methods in segregating the photometry of the disk galaxies by their inclination. The inclination of the galaxies therefore *impacts* their observed photometry, that in turn can be *learned* by a machine learning approach. We investigate how the inclination of galaxies impacts their observed photometry in the next section.

[4]In this work the biases are given in one significant figure and the RMS's in three decimal places, both $\pm$ one-sigma uncertainty.

3.3.2 Variance in Photometry due to Inclination of Disk Galaxies

Next, we seek to understand why the photo-z error correlates with the inclination of the photometry of the disk galaxies. Our approach is to establish the variance in the galaxy sample and its relation to the parameter(s) of interest, or the inclination of the galaxies in the current context. The Principal Component Analysis (PCA), which is adopted here, was shown to be a powerful technique for this purpose [e.g., 43, 64]. PCA identifies directions (or eigenvectors) in a multi-dimensional data space as such they represent the maximized sample variance. The lower the order of eigenvector, in this case the *eigenspectrum* [20], the more sample variance it describes. After relating the sample variance with the inclination, if possible, we can examine the involved eigenspectra to explain why in the edge-on galaxies the photo-z error is larger.

The first two eigenspectra calculated based on the photometry of the disk galaxies are shown in Figure 3.2. The 1st eigenspectrum resembles the mean spectrum of the galaxies. Perhaps more interestingly, the 2nd eigenspectrum visually resembles the extinction curve obtained in Paper I, despite the fact that they are obtained by two completely different approaches (PCA vs. composite spectra construction) and datasets (the photometry vs. the spectroscopy of the galaxy sample). The discrepancy between the 2nd eigenspectrum and the actual extinction curve is expected to be primarily due to variance in galaxy type within our disk galaxy sample.

To confirm that the 2nd mode represents the inclination effect to the photometry, we examine the distribution of the eigencoefficients of various orders as a function of the inclination (Figure 3.3). The eigencoefficients of a galaxy are the expansion coefficients of its photometry onto the eigenspectra. A clear separation is seen in the distribution of the 2nd eigencoefficients (a_2) between the face-on and edge-on galaxies. This separation is not seen, or as prominent, in the other orders of eigencoefficients. Since the large photo-z error occurs in edge-on galaxies, or, as inferred from Figure 3.3, for galaxies with negatively large a_2. In other words, the 2nd eigencoefficient is an *indicator* for the inclination of the disk galaxies.

Going back to Figure 3.2 to examine the eigenspectra, we see that the 1st eigenspectrum minus 2nd eigenspectrum results in a spectral energy distribution that is *redder* then the first eigenspectrum, or the average galaxy spectrum. If the adopted theoretical model in the template-based photo-z does not take account of this reddening effect in the edge-on galaxies, the model would need to be shifted to a higher-than-true redshift in order to match the redder colors of the galaxies. This situation results in an over-estimation of the photo-z, or what is seen in Figure 3.1a.

3.4 Corrections against the Inclination Effect

We discuss approaches to correct various properties of the disk galaxies against the inclination effect. The parameters considered are the restframe magnitudes, the flux density in an arbitrary stellar population model for the galaxies, and the photo-z.

3.4.1 On Restframe SDSS Magnitudes

We derive the following formulae for correcting restframe magnitudes of the whole disk galaxies in the SDSS u, g, r, i, z bands

$$M_u(1) = M_u(b/a) - 1.14 \cdot log_{10}^2(b/a) \ , \quad (3.1)$$

$$M_g(1) = M_g(b/a) - 0.76 \cdot log_{10}^2(b/a) \ , \quad (3.2)$$

$$M_r(1) = M_r(b/a) - 0.39 \cdot log_{10}^2(b/a) \ , \quad (3.3)$$

$$M_i(1) = M_i(b/a) - 0.17 \cdot log_{10}^2(b/a) \ , \quad (3.4)$$

$$M_z(1) = M_z(b/a) - 0.00 \cdot log_{10}^2(b/a) \ . \quad (3.5)$$

The underlying calculation is similar to that in Paper I, as such we fit to the relative extinction vs. b/a data the following relation

$$M(b/a) - M(1) = \eta \, log_{10}^2(b/a) \ , \tag{3.6}$$

where $M(b/a)$ is the K-corrected absolute magnitude of a galaxy at a given inclination. This functional form is taken to be the same for all of the SDSS bands, and the proportional constant η is fitted for each band. The left-hand side of Eqn. 3.6 is the relative extinction because $M(b/a) - M(1) = A^{\rm intrinsic}(b/a) - A^{\rm intrinsic}(1)$ (see Appendix 3.6 for details). The actual relative extinction vs. b/a values are given in Table 3.2. The magnitudes of the whole galaxies are considered in Eqn. 3.1–3.5, instead of the central 3″-diameter area of the galaxies that were considered in Paper I[5]. In particular, the model magnitudes ("modelMag") from the SDSS are used because they give unbiased colors of galaxies, a result of the flux being measured through equivalent apertures in all bands [54]. The one-sigma uncertainty for the best-fit η are, respectively, 0.05, 0.03, 0.02, 0.01 in u, g, r, and i. The corresponding reduced chi-square are 3.04, 2.79, 1.10, 1.18. The points for the relative extinction $z(b/a) - z(1)$ vs. inclination are scattered around zero and do not suggest any non-trivial functional form. We therefore do not attempt to fit the above relation in the z band, and assign zero to the proportional constant (Eqn. 3.5).

For the purpose of photo-z estimation, we will show in §3.4.3 that the color corrections derived from Eqn. 3.1–3.5 perform well, in the sense that the resultant photo-z are unbiased with inclination. On the other hand, the larger chi-squares in the u and

[5]We used the magnitudes derived from convolving the spectra with filters in the SDSS.

g bands suggest that the chosen relation may not be ideal. We therefore encourage the interpolation to the actual data listed in Table 3.2 when higher-accuracy corrections are required. We choose Eqn. 3.6 for the purpose of a direct comparison with literature [e.g., 58, 63], in which the powers of $\log_{10}(b/a)$ have been considered. Only even integers are allowed in the power index because $\log_{10}(b/a)$ is negative for all b/a values except unity. A power index of 4 is confirmed to provide a bad fit to our data, and a power index of 0 contradicts the data because it gives no b/a dependence. We plan to find other functional forms that may be unconventional but better describe the data.

The restframe $M_u(b/a) - M_g(b/a)$ vs. $M_g(b/a) - M_r(b/a)$ color-color diagram of our disk galaxies is shown in Figure 3.4, before and after the above inclination-dependent magnitude corrections. The average and the one-sigma sample scatter of the colors of the edge-on galaxies are, before the corrections: 1.37 ± 0.25 (for color $M_u(b/a) - M_g(b/a)$), 0.57 ± 0.14 ($M_g(b/a) - M_r(b/a)$), and after the corrections: 1.14 ± 0.26 ($M_u(b/a) - M_g(b/a)$), 0.35 ± 0.14 ($M_g(b/a) - M_r(b/a)$). The before-and-after color offset is ≈ 0.2 for both the $M_u(b/a) - M_g(b/a)$ and $M_g(b/a) - M_r(b/a)$ colors. Obviously, the colors of the face-on galaxies remain unchanged: 1.11 ± 0.12 ($M_u(b/a) - M_g(b/a)$) and 0.39 ± 0.08 ($M_g(b/a) - M_r(b/a)$). For both colors, the offset in the systematic locations between the edge-on galaxies and the face-on ones are greatly reduced after the corrections.

The 2nd-order power dependence of the relative extinction of the whole galaxies

on $log_{10}(b/a)$ agrees with that obtained by Unterborn and Ryden [58]. For the center of the disk galaxies (within 0.5 half-light radius), however, the extinction-inclination relation is steeper than a $log^2_{10}(b/a)$ dependence (Paper I). The difference likely reflects a higher extinction in the center relative to the edge of the galaxies, or an extinction radial gradient. We will investigate this finding in a separate paper.

3.4.2 On Flux Density of Stellar Population Models

The determination of many properties of galaxies, including the photo-z, involves fitting to the observational data a theoretical stellar population model. The model is defined by the related physical parameters, such as the stellar age and metallicity, at the correct amplitudes. In this kind of analysis, instead of correcting the observational data against the inclination effect as discussed previously, one can correct the theoretical model itself. The latter approach is at an expense of (or/and has the merit of) introducing the inclination of a galaxy as an extra parameter, which is to be determined simultaneously with the other properties during the minimization. Given a theoretical spectrum from a stellar population model, $f_\lambda(b/a = 1)$, its inclined flux densities can be calculated as follows

$$f_\lambda(b/a) = f_\lambda(1) \cdot s_\lambda(b/a) \ , \tag{3.7}$$

where

$$s_\lambda(b/a) = 10^{-0.4\,\eta_\lambda\,\log_{10}^4(b/a)} \tag{3.8}$$

is derived from the extinction curve of the disk galaxies and its variation with inclination as given in Paper I. In which,

$$\eta_\lambda = \sum_{j=0}^{3} \frac{a_j\,\tilde{\nu}^j}{\log_{10}^4\left(b/a|_{\rm ref}\right)}\ , \tag{3.9}$$

where $b/a|_{\rm ref}$ is a reference inclination. The wave number $\tilde{\nu}$ is the inverse of wavelength, in the unit of inverse micron, YOYOYO^{-1}. The coefficients a_j are listed in Table 4 of Paper I, where $b/a|_{\rm ref} = 0.17$. In presenting this formalism we use the extinction curve and its inclination dependence from Paper I, which apply to the inner 0.5 half-light radius of the disk galaxies. Extinction curves that are applicable to other parts of the galaxies, e.g. the whole galaxies, naturally can be used when required.

3.4.3 Photo-z from Random Forest Machine Learning

As presented above the machine learning approaches give photo-z which do not show prominent bias with inclination. This result is not entirely surprising, if it is

seen as the success of the methods in segregating the photometry of the disk galaxies by their inclination (see also §3.3). Here we consider the Random Forest approach, the power of which in estimating the photo-z is discussed in detail in Carliles et al. [16]. Basically this method builds an ensemble of randomized regression trees and computes regression estimates as the average of the individual regression estimates over those trees. The trees are built by recursively dividing the training set into a hierarchy of clusters of similar galaxies. The procedure minimizes the resubstitution error in the resultant clusters [Eqn. 1–3 of 16, and references therein]. We train a forest of 50 trees on 100,000 randomly selected galaxies from the SDSS spectroscopic sample, and regress on our disk galaxy sample to obtain the photo-z estimates shown in Figure 3.1(c).

Given the inclination of disk galaxies to be a parameter which modulates the variance in the photometric sample (§3.3.2), we deduce that the implicit inclusion of the inclination during the training procedure of the Random Forest method would give even better photo-z estimates than the case where only the SDSS colors are used (i.e., Figure 3.1(c)). Indeed, the photo-z estimates improve, with the resultant bias being 0.0004 and the RMS being 0.017. The error in the photo-z vs. inclination for this case is shown in Figure 3.5. It would be interesting to see if other machine learning approaches give similar improvement. For example, although the inclination of a galaxy was not included as a training parameter in the work by [**?**], their Artificial Neural Network approach in principal allows for multi-parameter training.

Another important question is whether the magnitude corrections (Eqn. 3.1–3.5) are applicable to deriving photo-z that are unbiased with inclination. Using the Random Forest approach, we select from the above training sample *face-on only* ($b/a = 0.9 - 1.0$) galaxies as our new training sample (about 50,000 objects). We train on the uncorrected u-g, g-r, r-i, i-z colors of this new sample, and regress on the corrected colors of our disk galaxy sample. The color corrections are done using Eqn. 3.1–3.5 (see Appendix 3.6). If the corrections give correct face-on colors of the disk galaxies, these colors should be fully described by those of the face-on galaxies in the training sample, and the resultant photo-z should be unbiased with inclination. Indeed, we find no inclination dependency in the photo-z error, as shown in Figure 3.6. The bias and RMS are respectively 0.002 ± 0.0003 and 0.018 ± 0.0005.

3.5 Conclusions

The reddening in the spectral energy distribution of a disk galaxy caused by its inclination, if not taken into account, impacts the accuracy of the derived photo-z. We present several approaches to correct the respective property of disk galaxies against the inclination effect. The considered properties are the restframe magnitudes, the flux densities of an arbitrary stellar population model for the disk galaxies, and the photo-z. We evaluate the performance of the inclination-dependent color corrections by using the accuracy of photo-z as a diagnostics, and find that the corrections give

statistically correct face-on colors of the disk galaxies.

We identify the inclination of the disk galaxies to be represented by a low order PCA mode of the sample, namely the 2nd mode. The inclination therefore modulates significantly the variance in the photometric sample. By considering the first two eigenspectra, the variance is revealed to be related to the reddening effect on the spectral energy distribution. The reddening effect leads to the aforementioned large photo-z error.

3.6 Magnitude & Color of Inclined Galaxies

The true absolute magnitude, M, of a totally transparent galaxy at any inclination is related to its apparent magnitude, m, as follows

$$m - M = 5 \log_{10}(d) - 5 + A^{\text{extrinsic}} + K \ , \tag{3.10}$$

where d, $A^{\text{extrinsic}}$, K are respectively the luminosity distance of the galaxy in parsecs, the extrinsic extinction (e.g., the sum of the Galactic and intergalactic extinctions), and the K-correction. We extend this formula to apply to a circular, dusty disk galaxy at an arbitrary inclination, as follows

$$m(b/a) - M(b/a) = 5 \log_{10}(d) - 5 + A^{\text{extrinsic}} + K(b/a) \ . \tag{3.11}$$

The extinction intrinsic to the galaxy is composed of two terms, namely, the inclination-independent and -dependent extinctions $A'^{\text{intrinsic}}$ and $A^{\text{intrinsic}}(b/a)$. They are related to the inclination-dependent absolute magnitude as follows

$$M(b/a) = M + A'^{\text{intrinsic}} + A^{\text{intrinsic}}(b/a) \ . \tag{3.12}$$

Combining Eqn. 3.12 and 3.11, we get

$$m(b/a) - M = 5\ \log_{10}(d) - 5 + A^{\rm extrinsic} + A'^{\rm intrinsic} + A^{\rm intrinsic}(b/a) + K(b/a)\ . \quad (3.13)$$

We derive from Eqn. 3.13 the relation between the face-on and inclined colors, for the u, g bands here and similarly for the other bands, to be

$$\begin{aligned} m_u(1) - m_g(1) \ &= \ m_u(b/a) - m_g(b/a) - [F_u(b/a) - F_g(b/a)] \\ &\quad - [K_u(b/a) - K_u(1)] \\ &\quad + [K_g(b/a) - K_g(1)]\ . \end{aligned} \quad (3.14)$$

The relative extinction is represented by a function of inclination $F(b/a)$, so that $M(b/a) - M(1) = A^{\rm intrinsic}(b/a) - A^{\rm intrinsic}(1) = F(b/a)$. The choice of $F(b/a)$ in this work is given in Eqn. 3.6 of §3.4. In the application of photo-z estimation, the K-correction terms are unknown a priori because the spectroscopic redshift of the galaxy in question is unknown. A focus of this work, however, is not the photo-z amplitude but the dependency of photo-z error on the inclination. Since our disk galaxies are local, the K-corrections are only higher-order modulations to their colors and hence to the photo-z error. We therefore neglect the K-correction terms in Eqn. 3.14 and adopt $F_u(b/a) - F_g(b/a)$ (and similarly for the other colors) as the color corrections in the Random Forest case study in §3.4.3. We plan to explore an iterative approach

to simultaneously estimate both color corrections and K-corrections in the future.

Table 3.1 Statistics of SDSS photo-z error.

b/a[a]	number	mean[b]	median	sigma	relative bias[c]
0.17 ± 0.024	272	0.021	0.020	0.038	0.036
0.25 ± 0.028	829	0.014	0.013	0.034	0.028
0.35 ± 0.028	942	0.006	0.005	0.041	0.021
0.45 ± 0.028	806	-0.004	-0.009	0.032	0.011
0.55 ± 0.029	783	-0.008	-0.015	0.030	0.007
0.65 ± 0.030	729	-0.011	-0.017	0.032	0.004
0.75 ± 0.029	736	-0.012	-0.019	0.030	0.003
0.85 ± 0.030	759	-0.013	-0.021	0.032	0.001
0.94 ± 0.026	429	-0.015	-0.021	0.028	0.000

Note. — The photo-z referred here were derived using the template-fitting approach [24].

[a]The mean $\pm$ one-sigma sample scatter of the apparent minor to major axis ratio in the sample of disk galaxies.

[b]The ensemble bias is equal to -0.004 ± 0.001, or close to zero. Therefore, the mean photo-z error in each inclination bin is effectively the bias in the inclined galaxies relative to that in the full sample.

[c]The photo-z bias in inclined galaxies relative to that in face-on galaxies.

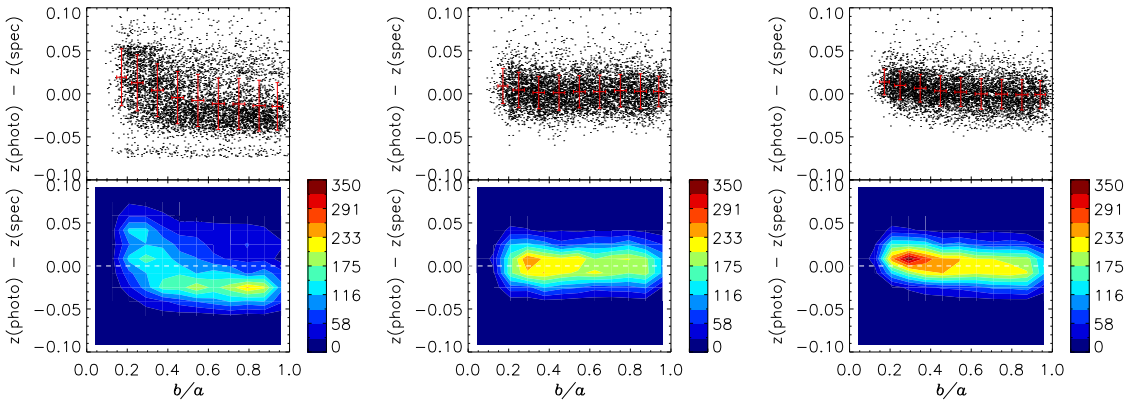
Table 3.2 Relative extinction as a function of inclination of whole disk galaxies.

b/a[a]	$M_u(b/a) - M_u(1)$[b]	$M_g(b/a) - M_g(1)$	$M_r(b/a) - M_r(1)$	$M_i(b/a) - M_i(1)$	$M_z(b/a) - M_z(1)$
0.17 ± 0.0015	0.62 ± 0.03	0.41 ± 0.02	0.21 ± 0.03	0.10 ± 0.03	-0.04 ± 0.03
0.25 ± 0.0010	0.46 ± 0.01	0.31 ± 0.02	0.16 ± 0.02	0.08 ± 0.02	-0.04 ± 0.02
0.35 ± 0.0009	0.27 ± 0.02	0.16 ± 0.02	0.06 ± 0.02	-0.00 ± 0.02	-0.07 ± 0.02
0.45 ± 0.0010	0.17 ± 0.02	0.13 ± 0.02	0.07 ± 0.02	0.02 ± 0.02	-0.03 ± 0.02
0.55 ± 0.0010	0.11 ± 0.02	0.11 ± 0.02	0.06 ± 0.02	0.05 ± 0.02	0.01 ± 0.03
0.65 ± 0.0011	0.03 ± 0.02	0.04 ± 0.02	0.01 ± 0.02	-0.00 ± 0.02	-0.02 ± 0.02
0.75 ± 0.0011	0.04 ± 0.02	0.05 ± 0.02	0.03 ± 0.02	0.02 ± 0.02	0.00 ± 0.03
0.94 ± 0.0012	0.00 ± 0.03	0.00 ± 0.03	0.00 ± 0.03	0.00 ± 0.03	0.00 ± 0.03

Note. — The SDSS model magnitudes are considered here.

[a]The mean $\pm$ one standard deviation of the mean (SDOM) of the apparent minor to major axis ratio in the sample of disk galaxies. The number of galaxies in each inclination bin are listed in Table 3.1.

[b]The mean $\pm$ one SDOM of the relative extinction.



(a) The photo-z based on the template-fitting approach.

(b) The "CC1" photo-z based on the Artificial Neural Network approach.

(c) The photo-z based on the Random Forest approach.

Figure 3.1 The photo-z error as a function of the inclination of the disk galaxies. The top panels are scatter plots, with the error bar represents the mean $\pm$ one-sigma sample scatter of the binned data. The bottom panels show the corresponding number contours, smoothed over a 12×12 grid within the shown plotting ranges. The photo-z shown in Figure 3.1(a) and 3.1(b) are taken from the SDSS DR6. The Random Forest photo-z are calculated in this work, shown in Figure 3.1(c).

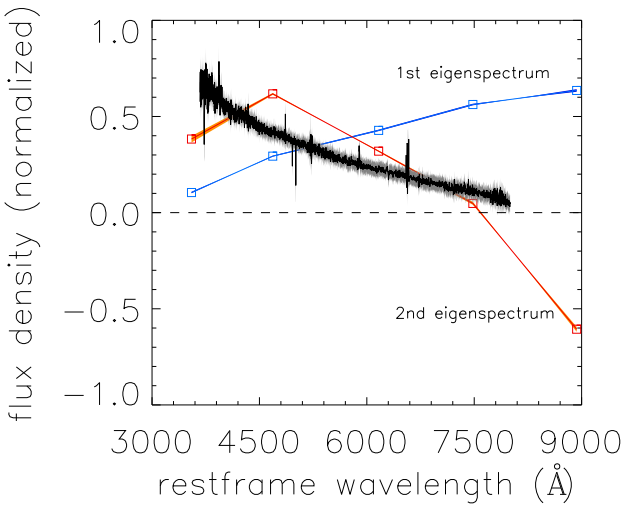


Figure 3.2 The 1st (blue) and 2nd (red) eigenspectra constructed from the photometry of our disk galaxy sample. The extinction curve from Paper I is plotted for comparison, in black line. The 2nd eigenspectrum resembles the extinction curve.

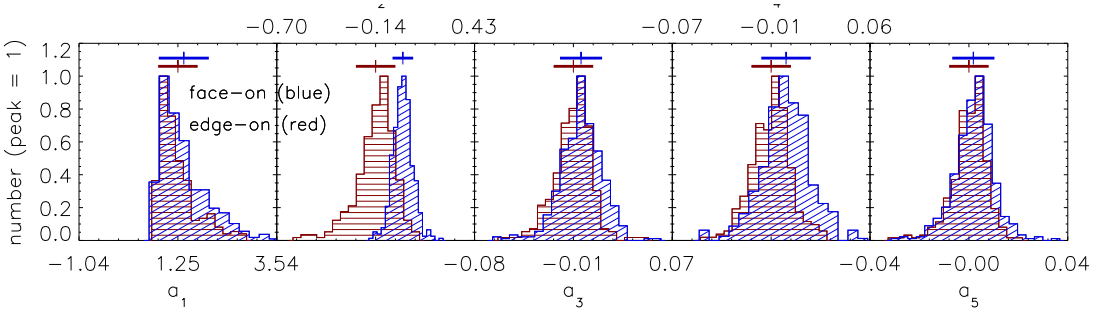


Figure 3.3 The comparison of the distribution of the eigencoefficients between the face-on (blue) and edge-on (red) galaxies, from the first (a_1) to the fifth (a_5, or the last) modes in a Principal Component Analysis (PCA). Among all of the PCA modes, the variance in the photometry of the disk galaxies due to their inclination is best described by the 2nd mode.

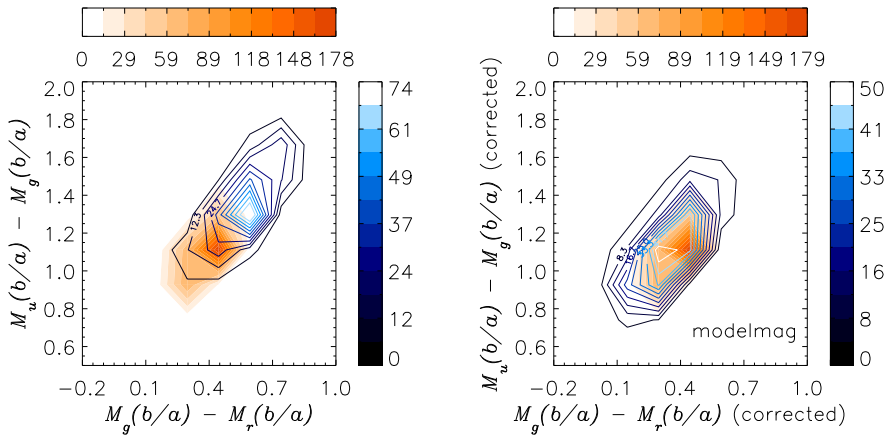


Figure 3.4 The restframe $M_u(b/a) - M_g(b/a)$ vs. $M_g(b/a) - M_r(b/a)$ diagram of our disk galaxy sample, before (left) and after (right) the inclination correction. The face-on galaxies are represented by the filled contours, whereas the edge-on ones by the line contours.

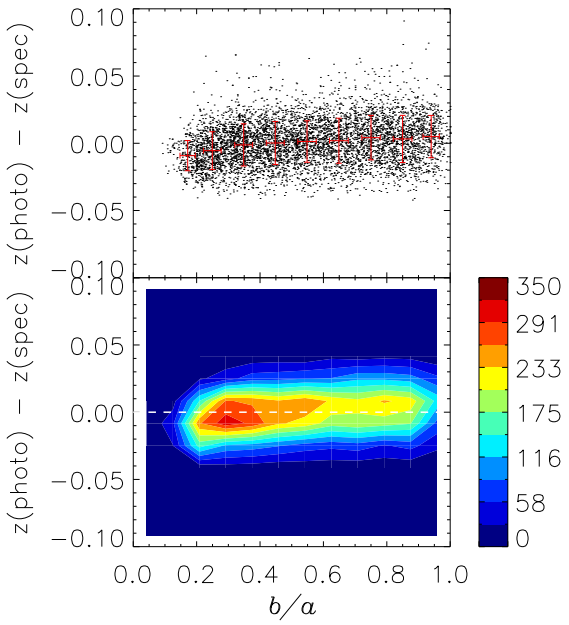


Figure 3.5 The photo-z error as a function of the inclination of the disk galaxies, using the Random Forest approach. In the training procedure the inclinations of the galaxies are included in addition to their 4 SDSS colors. Compared with Figure 3.1c in which only the SDSS colors are included in the training, the bias is reduced.

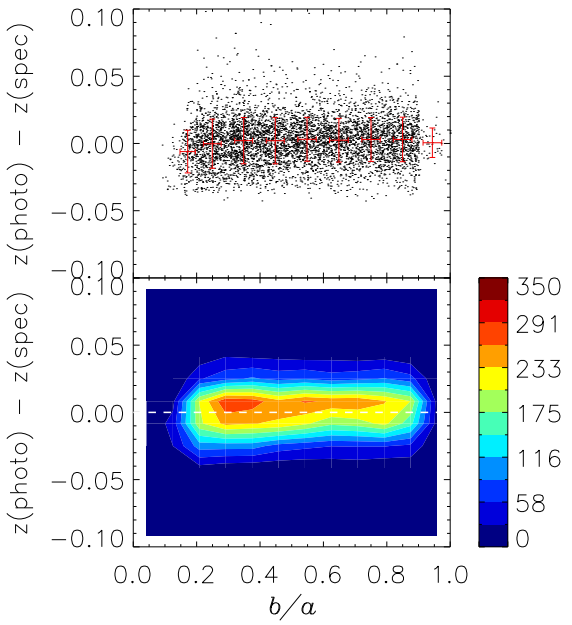


Figure 3.6 The photo-z error as a function of the inclination of the disk galaxies, using the Random Forest approach. The training is performed on the uncorrected u-g, g-r, r-i, i-z colors of a random sample of face-on only galaxies, and the regression is performed on the corrected u-g, g-r, r-i, i-z of our disk galaxy sample, corrected through Eqn. 3.1–3.5. No inclination dependency is present in the photo-z error, implying that the color corrections give statistically correct face-on colors of the disk galaxies.

Chapter 4

A C# Implementation of Random Forest Regression

4.1 Motivation

In Carliles et al. [16] we describe a process for estimating galaxy redshifts with confidence intervals using Random Forests [10, 38] trained on photometric data from the sixth data release of the Sloan Digital Sky Survey [3]. For this work we used the randomForest package version 4.5-18 [42] in the R statistical data analysis environment version 2.5.1 [1]. While this package and environment are extraordinarily useful for exploratory data analysis on small data sets, we encountered some difficulty when attempting to scale the analysis up to larger data sets. To address

[1] `http://www.r-project.org/`

this difficulty, and with an eye toward investigating and refining the methodology and integrating with Microsoft SQL Server, we chose to write our own implementation in C# .NET. The source code for our implementation is publicly available at `https://bitbucket.org/SamuelCarliles/randomforests`.

4.2 Implementation notes

4.2.1 Overview of the model

We compose forests naïvely of independently trained regression trees; when training a forest of size $|T|$, we simply schedule $|T|$ separate tree-training tasks and rely on the .NET Framework's default thread scheduler to manage CPU utilization. We encode regression trees naïvely as binary trees, and we utilize a stack-based depth-first training procedure in which left branches take precedence.

4.2.2 Time complexity

Computing the best split in a single dimension

At a given node, we are given a set of N_P training objects $\mathcal{X} = \{(X_i, Y_i)\}$, $i \in \{1, \ldots, N_P\}$ where the X_i are k-dimensional real input vectors, and the Y_i are scalar real response values. The set-valued functions $\mathcal{X}_L(d, s) = \{(X_i, Y_i) \in \mathcal{X} : X_{id} \leq s\}$ and $\mathcal{X}_R(d, s) = \mathcal{X} - \mathcal{X}_L(d, s)$ define a partition of $\mathcal{X}$ into disjoint subsets split along

dimension d at point s. Henceforth, context permitting, we will abbreviate these as $\mathcal{X}_L$ and $\mathcal{X}_R$, respectively. The risk at $\mathcal{X}$ is defined as

$$\mathrm{R}(\mathcal{X}) = \mathrm{P}((X,Y) \in \mathcal{X}) \operatorname{Var}(Y|(X,Y) \in \mathcal{X}), \tag{4.1}$$

[11, pp. 228-232] and the gain in risk minimization is defined as

$$\Delta\,\mathrm{R}(\mathcal{X}, d, s) = \mathrm{R}(\mathcal{X}) - \mathrm{R}(\mathcal{X}_L) - \mathrm{R}(\mathcal{X}_R). \tag{4.2}$$

We seek to partition $\mathcal{X}$ by choosing a dimension $d^* \in \{1, \ldots, k\}$ and a split point $s^* \in \mathbb{R}$ such that equation 4.2 is maximized:

$$(d^*, s^*) = \underset{d \in \{1,\ldots,k\}, s \in \mathbb{R}}{\operatorname{argmax}} \Delta\,\mathrm{R}(\mathcal{X}, d, s). \tag{4.3}$$

A useful consequence of the Law of Total Probability is that

$$\mathrm{P}((X,Y) \in \mathcal{X}) = \mathrm{P}((X,Y) \in \mathcal{X}_L) + \mathrm{P}((X,Y) \in \mathcal{X}_R). \tag{4.4}$$

Since $\mathcal{X}_L$ and $\mathcal{X}_R$ form a partition of $\mathcal{X}$, the Law of Total Expectation applies:

$$\begin{aligned} & \mathrm{E}[h(Y)|(X,Y) \in \mathcal{X}]\,\mathrm{P}((X,Y) \in \mathcal{X}) \\ = \quad & \mathrm{E}[h(Y)|(X,Y) \in \mathcal{X}_L]\,\mathrm{P}((X,Y) \in \mathcal{X}_L) \\ + \quad & \mathrm{E}[h(Y)|(X,Y) \in \mathcal{X}_R]\,\mathrm{P}((X,Y) \in \mathcal{X}_R), \end{aligned} \tag{4.5}$$

where h is some integrable function of the response variable. Now with the common identity $\mathrm{Var}(X) = \mathrm{E}[X^2] - \mathrm{E}[X]^2$, the gain in risk minimization can be simplified:

$$\begin{aligned} \Delta\,\mathrm{R}(\mathcal{X}, d, s) \quad = \quad & \mathrm{Var}(Y|(X,Y) \in \mathcal{X})\,\mathrm{P}((X,Y) \in \mathcal{X}) \\ - \quad & \mathrm{Var}(Y|(X,Y) \in \mathcal{X}_L)\,\mathrm{P}((X,Y) \in \mathcal{X}_L) \\ - \quad & \mathrm{Var}(Y|(X,Y) \in \mathcal{X}_R)\,\mathrm{P}((X,Y) \in \mathcal{X}_R) \\ = \quad & \mathrm{E}[Y|(X,Y) \in \mathcal{X}_L]^2\,\mathrm{P}((X,Y) \in \mathcal{X}_L) \\ + \quad & \mathrm{E}[Y|(X,Y) \in \mathcal{X}_R]^2\,\mathrm{P}((X,Y) \in \mathcal{X}_R) \\ - \quad & \mathrm{E}[Y|(X,Y) \in \mathcal{X}]^2\,\mathrm{P}((X,Y) \in \mathcal{X}). \end{aligned} \tag{4.6}$$

If we plug in maximum likelihood estimates of all terms, the empirical gain in risk minimization becomes

$$\begin{aligned}
\widehat{\Delta \mathrm{R}}(\mathcal{X}, d, s) &= \frac{N_{\mathcal{X}_L}}{N}\bar{Y}_{\mathcal{X}_L}^2 + \frac{N_{\mathcal{X}_R}}{N}\bar{Y}_{\mathcal{X}_R}^2 - \frac{N_{\mathcal{X}}}{N}\bar{Y}_{\mathcal{X}}^2 \\
&= \frac{1}{N}\left[\frac{(\sum_{\mathcal{X}_L} Y)^2}{N_{\mathcal{X}_L}} + \frac{(\sum_{\mathcal{X}_R} Y)^2}{N_{\mathcal{X}_R}} - \frac{(\sum_{\mathcal{X}} Y)^2}{N_{\mathcal{X}}}\right] \\
&= \frac{1}{N}\left[\frac{S_{\mathcal{X}_L}^2}{N_{\mathcal{X}_L}} + \frac{(S_{\mathcal{X}} - S_{\mathcal{X}_L})^2}{N_{\mathcal{X}} - N_{\mathcal{X}_L}} - \frac{S_{\mathcal{X}}^2}{N_{\mathcal{X}}}\right], \qquad (4.7)
\end{aligned}$$

where $N_{\mathcal{X}} = |\mathcal{X}|$, and $S_{\mathcal{X}} = \sum_{\mathcal{X}} Y$. Observe that $N_{\mathcal{X}}$ and $S_{\mathcal{X}}$ remain constant for all choices of d and s, and that $N_{\mathcal{X}_R}$ and $S_{\mathcal{X}_R}$ are expressed respectively in terms of $N_{\mathcal{X}}$ and $N_{\mathcal{X}_L}$, and $S_{\mathcal{X}}$ and $S_{\mathcal{X}_L}$. Thus we may sort $\mathcal{X}$ along dimension d, then simply iterate "left-to-right" among potential choices of s in the range of X values present in $\mathcal{X}$ in a single linear pass to determine a risk-minimizing value of s. Using the Array.Sort method in the .NET Framework, the average cost to sort is $\mathrm{O}(n \log n)$, and the linear pass to evaluate all potential choices of s does not increase the asymptotic cost.

Total asymptotic training time

Two tree structures which are useful to consider are the deepest possible, in which at each level we peel off m training observations, and a perfect binary tree, which is the shallowest possible. The deepest tree has a depth of $O(n)$ resulting in an average case training time much less than $O(n^2 \log n)$, while the shallowest has a depth of $\lg n$ resulting in an average case training time of $O(n \log^2 n)$.

4.2.3 Space complexity

Indexes on a stationary data array

We store a single copy of the original data set in an array D. Let us denote the forest as a set T of regression trees indexed by $t \in \{1, \ldots, |T|\}$. For each given tree T_t, we create an array of indices into D either with or without replacement as specified. Call this array B_t; B_t represents T_t's bootstrap sample drawn from D. We then treat each B_t as fixed, and we create an index into B_t, call it A_t. It is this index A_t which we pass to the root node of the tree. Now begins the training procedure proper – the root node is given what looks to it like a data array, but this apparent array is actually just an index into the bootstrap sample, which is in turn an index into the original data array. If the stopping criterion of maximal node size has not been met, the node selects its subspace of dimensions in which to consider splitting, and for each dimension, applies the procedure described in section 4.2.2. The splitting procedure entails sorting the data along each input dimension under consideration, but really the node just sorts its index using the appropriate dimension of the input data as a sorting key. When the split dimension d^* and split point s^* are chosen, the node splits its index into "left" and "right" indexes on the underlying data, instantiates left and right child nodes with these indexes, and pushes them onto left and right stacks, respectively, then proceeds to the next iteration of the training loop. No data is ever duplicated, but at each split we allocate enough memory to reference the elements in the left partition, and we don't currently shrink the right partition, but

simply pass it the parent array and update its range. In the worst case, we might always partition such that the right child gets the minimal leaf size, so that we always allocate the maximal number of addresses in the left partition. This results in $O(n^2)$ space complexity, though the average case is obviously much smaller since we have yet to consume more than a few GB RAM per tree in any trial when running in "fast" mode. In any case we will rectify this space inefficiency in a future release by shrinking the right partitions. $O(n)$ space complexity seems achievable. Our procedure is illustrated in figure 4.1.

4.2.4 Regression

Producing regression estimates on new observations is a straightforward traversal down the tree from root to terminal node. The natural tendency is to write a recursive traversal, but on large data sets, the trees quickly become sufficiently deep that a recursive implementation exhausts the call stack. Thus we use an iterative traversal.

4.3 Photometric Redshift Estimation

4.3.1 Photometric Redshifts

A popular application of regression techniques in the astronomical problem domain is that of estimating galaxy distances from Earth in units of redshift using photometric

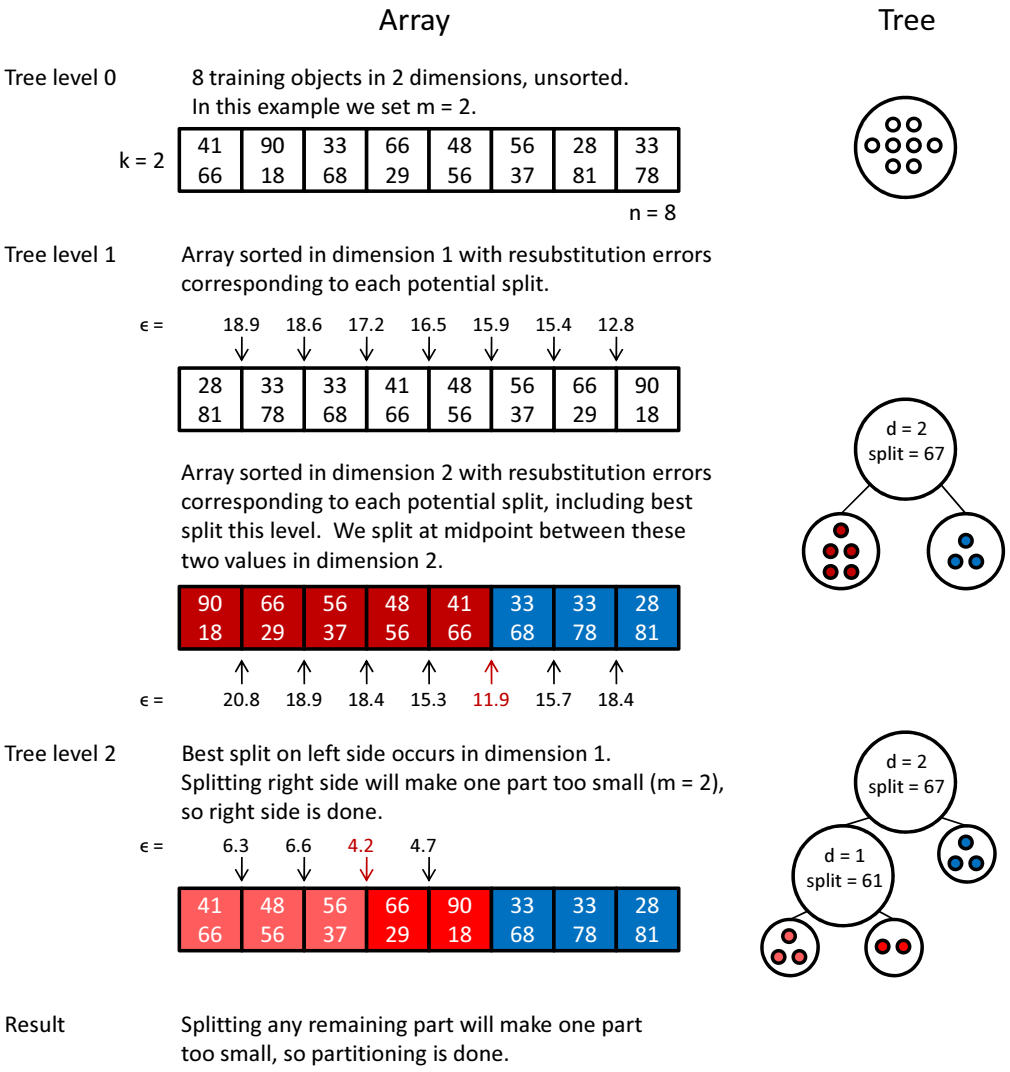


Figure 4.1 In-place partitioning of 8 training objects in 2 dimensions with no fewer than two objects in each leaf node. The splitting criterion is resubsitution error. Nodes within a "cluster" are shown sorted in place along the dimension with the best split, which is how we implement our algorithm. A binary tree structure is built with internal nodes storing optimal split dimension and split point, and leaf nodes (corresponding to the resulting light red, brighter red, and blue clusters of training objects in the last stage) storing pointers into the input array of training data objects. In fact we operate on an array of references into the input array, keeping the original array intact.

data [37]. Redshift is a real-valued measure of how far the wavelength of light emitted by a celestial object has shifted toward the red end of the electromagnetic spectrum due to the Doppler effect. We can measure galaxy redshift with high accuracy using spectroscopy, and spectroscopic redshifts can then be used as tags on observations for the purposes of supervised learning. An older version of the current package, along with another kd-tree-based technique were used to produce two sets of redshift estimates for the eigth data release [22] of the Sloan Digital Sky Survey [4]. We use a dataset derived from that training set for the present benchmarking procedure.

4.3.2 Tuning Random Forest Parameters

In practice, an important question with Random Forest regression is how large to make the forest. We should use as many trees as are necessary to converge to within a small epsilon of the limiting response estimates given the training data. We hypothesized that for a given test observation x, in the limit of infinite trees, each training observation would contribute to the response with its own constant frequency. These limiting frequencies may be regarded as weights in a discrete kernel such that the inner product of this kernel and the training response vector produce the regression estimate. For any particular Random Forest, the empirical frequencies of training observation contributions would be natural estimates for the limiting frequencies. We measure the *effective bandwidth* of the kernel used for test observation x as the inverse

sum of squared probabilities of occurrence taken over training observations:

$$\mathrm{h}(x) = \frac{1}{\sum_{i \in \{1,\ldots,N\}} p_i(x)^2}, \tag{4.8}$$

We estimate this with plugin estimates of $p_i(x)$:

$$\hat{p}_i(x) = \frac{1}{T} \sum_{t \in \{1,\ldots,T\}} \frac{N_{it}(x)}{N_t(x)}, \tag{4.9}$$

where T is the number of trees, $N_{it}(x)$ is the number of times training observation i contributes to test point x's regression estimate in tree t, and $N_t(x)$ is the number of contributions to x's regression estimate in tree t (including possible multiple occurrences of a single observation).

The bandwidth ranges over $\{1, \ldots, N\}$, and may be regarded as a count of how many distinct training observations contribute significantly to x's regression estimate using the given forest. Thus, when the bandwidth converges for x, this is an indication that the regression estimate for x has converged. Since the bandwidth is measured in units of distinct contributing training observations, a reasonable naïve stopping criterion for training would seem to be when adding several trees does not increase the bandwidth by some small threshold integral amount for any x' in the test set. In practice, however, we found that even in the "very large forest" regime, the band-

width could vary between successive trials by margins which would be difficult to call "small," though there did appear to be consistent bounds on the variance. It may be that the best we can do is to model the bandwidth as a random variable for which we seek convergence in distribution. In any case, given the apparent weakness of any pointwise convergence of our bandwidth function estimator, to construct our stopping criterion we instead observed the mean bandwidth over all observations in the test set; this appeared to converge nicely. Since our Random Forests train so quickly, it worked well enough for us simply to grow forests of size $T_t \in \{1, 10, 50, 100, \ldots\}$ sequentially and observe the convergence of the mean bandwidth. Interestingly, the RMS error on the test set appeared to converge somewhat more quickly than the mean bandwidth (figure 4.2.)

4.3.2.0.1

4.3.3 Benchmark Trials

We sought to evaluate performance of our implementation as functions of the two most important model parameters: training set sample size and forest size. When producing bandwidth estimates in the C# implementation, we use a different algorithm for the regression step which is somewhat slower (and vastly more spendthrifty with RAM) than the "fast" mode. This should be addressed in a future release. In the meantime, for all timed trials we ran the C# implementation in its "fast" mode.

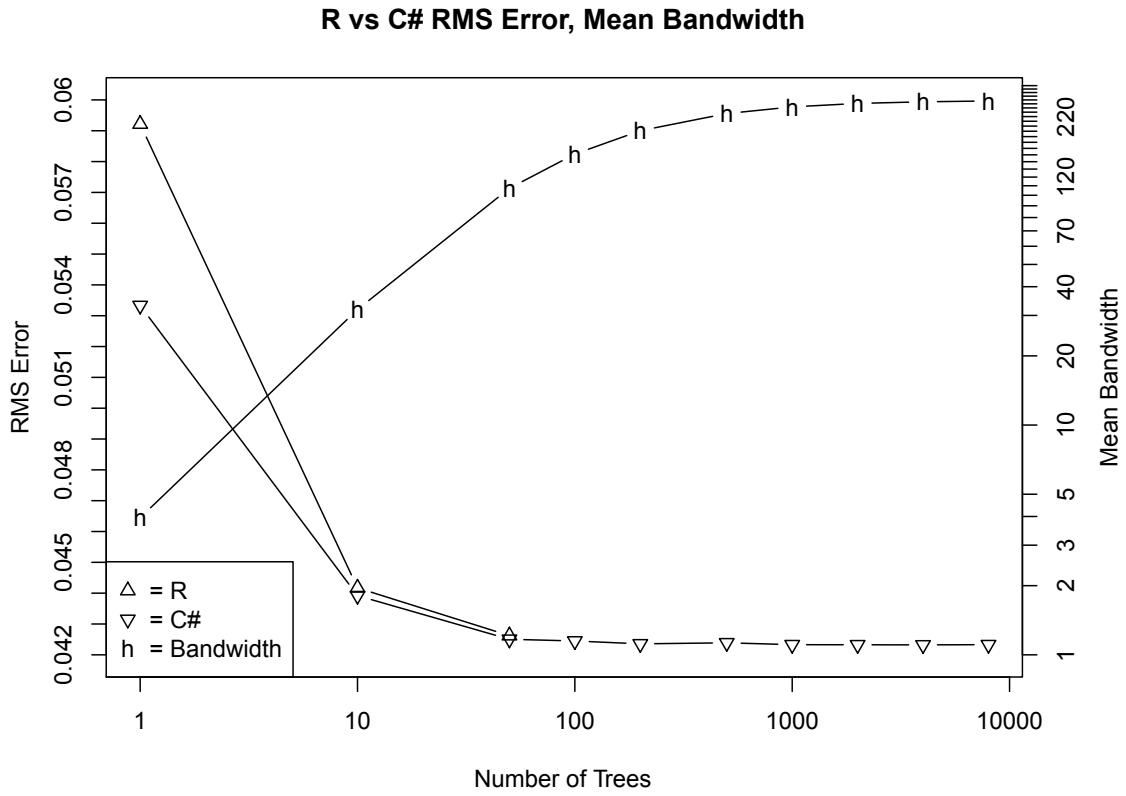


Figure 4.2 RMS error for forests of 1, 10, and 50 trees in R and C#, and additionally 100, 200, 500, 1000, 2000, 4000, and 8000 trees in C#. Observed mean effective bandwidth for these forest sizes in C#.

The data set consisted of spectroscopic redshifts as response tags, with inputs of photometric magnitude differences between five different wavelength bands, and an additional measure of the galaxies' "inclination," [65] which describes whether we're observing the galaxy more on its face or on its edge. The test machine ran Microsoft Windows 7 Enterprise 64-bit with Service Pack 1 on an Intel Core i7-4770K CPU running at 3.5GHz, with 32GB RAM. We used the randomForest package version 4.6-7 in R version 3.1.0 as a reference implementation to compare against, since this implementation is almost certainly the most commonly used, and since it's the one we use for day-to-day data analysis.

Execution Time versus Training Set Size

To see how the execution time scales with training set size, we ran 100 trials each training a single tree on training set sizes of 10^3, 10^4, 10^5, 2×10^5, 4×10^5, and 8×10^5. The test set size for all trials was approximately $75,000$. The sample means over these 100 trials at each size are presented in figure 4.3. For these trials we excluded the time to ingest data from the R trials, but included it for the C# trials out of implementational convenience. Naturally, as the execution time increases, the data ingest diminishes in significance. Using this data set, at training set sizes of 10^5 and beyond, the C# implementation yields a speedup starting at approximately $10\times$ and increasing to approximately $200\times$ on the largest training set size tested, with the gain trending increasingly large as the training set size increases; it appears to be

asymptotically faster.

RMS Error and Execution Time versus Forest Size

To observe both the convergence in RMS error and the scale of execution time as forest size increases, we trained on 8×10^5 observations, and set aside approximately $75,000$ observations as a test set. We first ran trials using only the C# implementation in order to determine a suitable upper limit on forest size, knowing that we could do so quickly with our implementation. Even using RMS error as a stopping criterion, there appears to be a benefit to training at least as many as $2,000$ trees. We trained as many as $8,000$ trees before concluding that the mean bandwidth had converged more or less, and that all information in the dataset had been incorporated into the regression estimates. As can be seen in figure 4.2, both versions appear to converge to the same limiting RMS error. In total for the C# implementation we ran 10 single-tree trials, and a single trial for forests of size 10; 50; 100; 200; 500; $1,000$; $2,000$; $4,000$; and $8,000$ since larger forests are intrinsically equivalent to running multiple trials in batch. We then ran trials using the R implementation, sufficient to project expected ideal running times for all forest sizes in the test suite. The test machine had four physical cores with Intel Hyperthreading, for an apparent eight cores. The randomForest package in R seems not to have multiprocessor support, as the total elapsed execution time scaled almost perfectly linearly with the number of trees. We ran 10 single-tree trials and one trial each for forests of size 10 and 50. The R package

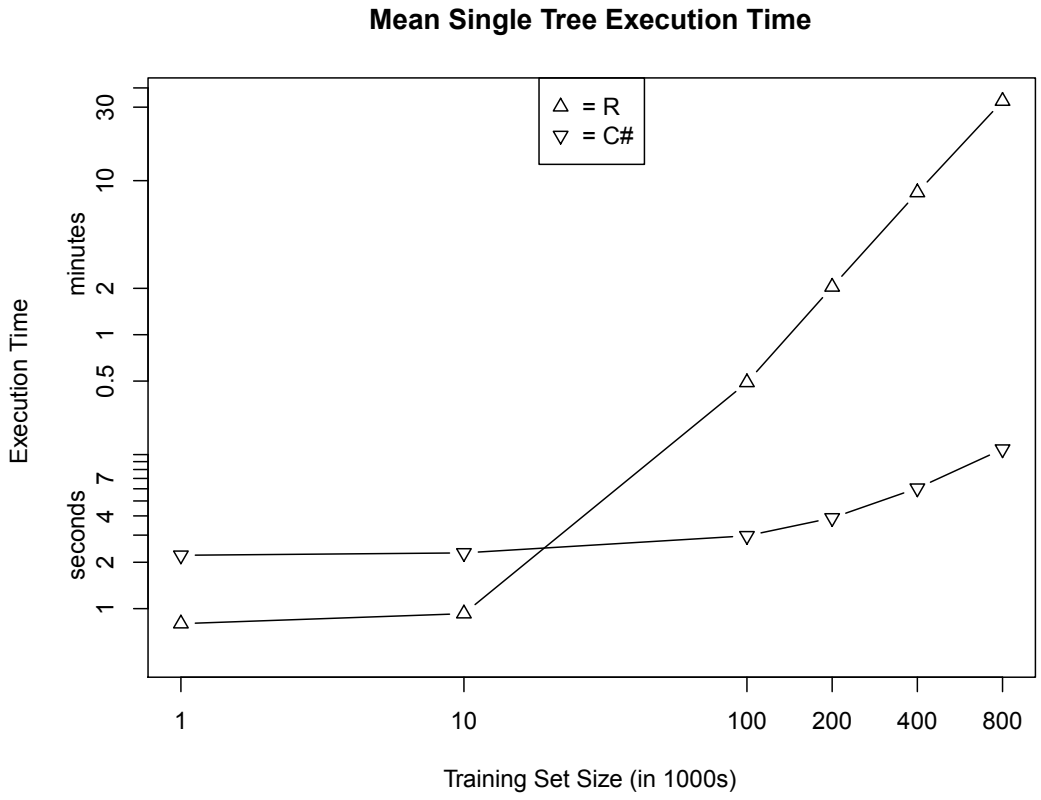


Figure 4.3 Mean training and regression execution time for a single tree in R and C# with training set sizes of $1k$, $10k$, $100k$, $200k$, $400k$, and $800k$. The mean was computed over 100 trials at each training set size. Out of scripting convenience, R values do not include time to ingest data, while the C# values do. Both axes are plotted in log scale, with minute labels at the equivalent second ticks for ease of interpretation.

consistently took about 33 minutes per tree to train and regress on the training and test sets, regardless of the specified forest size. Since the per-tree execution time for R was so consistent in trials up to forests of 50 trees, we forewent the larger forest trials and instead projected execution times assuming the existence of an "ideal" scheduler to handle the task of distributing smaller forest R jobs across multiple processors. For single-tree trials, the values presented are averaged over the 10 trials since that option was available to us. On the test machine, the approximately 200× speedup using the C# package holds for all forest sizes, doing in just over 6 hours what would take the R package over 3 weeks to do under ideal conditions (see figure 4.4.)

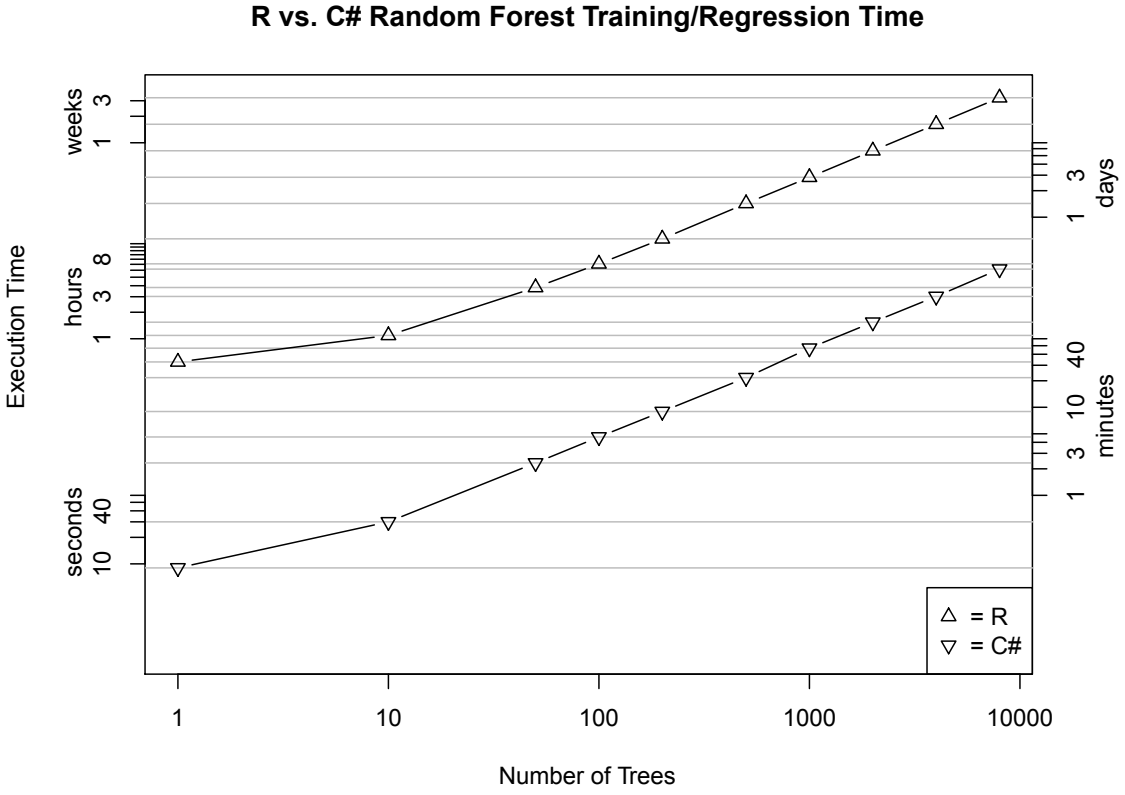


Figure 4.4 Training and regression execution time for forests of 1, 10, 50, 100, 200, 500, 1000, 2000, 4000, and 8000 trees in R and C#. Since the R version is not intrinsically multi-threaded, sequential per-tree execution times were measured on forests of 1, 10, and 50 trees in R, and were uniformly approximately 33 minutes per tree. The results reported for R are projections using this per-tree execution time and assuming an ideal process scheduler and "frictionless" multi-threading on the same machine. C# execution times reported are actual. For this plot, data ingest time was included for both R and C# trials out of scripting convenience. Again for ease of interpretation, both axes are plotted in log scale with all values in seconds, and with more useful labels at equivalent second ticks.

Chapter 5

A CUDA Implementation of Random Forest Regression

5.1 Motivation

Scientific data sets are approaching petabytes today. At the same time, enterprise data warehouses routinely store and process even larger amounts of data. Most of the analyses performed over these datasets (e.g., data mining, regressions, calculating aggregates and statistics, etc.) need to look at large fractions of the stored data. Thereby, sequential throughput is becoming the most relevant metric to measure the performance of data-intensive systems. Given that the relevant data sets do not fit in main memory, they have to be stored and retrieved from disks. Szalay et al. [57] demonstrate a very low power cluster with fast I/O. The general idea behind this clus-

ter is to correct a shortcoming of traditional clusters which becomes an acute problem when trying to scale up the data set size. Such clusters are composed of commodity PCs utilizing extremely powerful CPUs mounted on mainboards capable of accommodating disk controllers sufficient to address vast storage capacity. However, typical commodity PC architecture is constrained by relatively narrow I/O busses between disk controllers and CPUs. In contrast, newer low power systems with relatively weak CPU processing capability supplemented by sophisticated graphics processors, and with higher I/O bandwidth relative to CPU clock speed seem the ideal components with which to build more a "balanced" cluster to accommodate data-intensive scientific computing pipelines. Put simply, we believe we can waste less power on idle CPU cycles and disk rotations. This, however, will require an altered approach to computation – one suited better for this scaled-down-and-out approach to system design. Our Random Forest implementation is intended to address this need to do fast data analysis on enormous data sets in such a scalable, low power cluster environment, taking advantage of the ability to do General Purpose Graphics Processing Unit (GPGPU) programming on these cluster nodes. No known available Random Forest implementation addresses this need.

5.2 Clusters

In order to evaluate our new implementation, we compare the performance of this new highly parallel implementation of Random Forest regression running on this new low power cluster with our serial implementation running on a more traditional high power benchmark cluster. We now describe both systems in more detail. As a basis for comparison, we consider three properties established by Gene Amdahl to characterize a well-balanced computer system:

1. One bit of sequential I/O per second per instruction per second (the Amdahl Number).

2. Memory with a MB/MIPS ratio close to 1 (the Amdahl memory ratio).

3. One I/O operation per 50,000 instructions (the Amdahl IOPS ratio).

5.2.1 The Benchmark System - GrayWulf Cluster

The GrayWulf system [56] represents a state-of-the-art architecture for data-intensive applications. Focusing primarily on sequential I/O performance, each GrayWulf server consists of two Dell MD1000 storage chassis containing 30 locally attached 750GB SATA drives, connected to two Dell PERC/6 controllers in a Dell 2950 server with 24GB of memory and two four-core Intel Xeon processors clocked at 2.66GHz. The raw read performance of this system is 1.5GB/s, translating to 15,000 seconds

(4.2 hours) to read all the disks. Such a building block costs approximately $12,000 in 2009 prices and offers a total storage capacity of 22.5TB. Its power consumption is 1,150W. It has an Amdahl number of 0.56 and a memory ratio of 1.12 MB/MIPS. The third Amdahl law requires 426 KIOPS to match the CPU speed, while the hard disks can only deliver about 6 KIOPS, a ratio of 0.014.

The GrayWulf cluster consists of 50 such servers, and this parallelism linearly increases the aggregate bandwidth to 75GB/sec, the total amount of storage to more than 1.1 PB, and the power consumption to 56 kW. Doubling the storage capacity of the GrayWulf cluster, while maintaining its per-node current throughput, would require using twice as many servers, thereby doubling its power consumption. Alternatively, one could divide the same amount of data over twice as many disks (and servers) to double the system's throughput, again at the cost of doubling its power consumption. At this rate, the cost of building and operating these ever expanding facilities is becoming a major roadblock not only for universities, but even for large corporations [35]. Thus tackling the next generation of data-intensive computations in a power-efficient fashion requires a radical departure from existing approaches.

One can extend the Amdahl number from hardware platforms to computational problems: take the data set size in bits and divide with the number of cycles required to process it. While supercomputer simulations have Amdahl numbers of 10^{-5}, the Amdahl numbers of pipeline processing of observational astronomy data increase to 10^{-2}, and user analyses of derived catalogs and database queries approach unity. Thus, aiming for systems with high Amdahl numbers at a given performance level is likely to result in balanced and thus energy-efficient systems.

5.2.2 The Challenger - Amdahl Cluster

This cluster was designed specifically to satisfy the three Amdahl properties described above. Rather than increasing the number of disks, we attempt to increase the per-disk throughput, thereby decreasing the total number of servers, ideally while keeping per-disk power consumption low. In fact, Solid State Disks (SSDs) that use similar flash memory as the one used in memory cards, provide both desired features. Current SSDs offer sequential I/O throughput of 90-250 MB/s and 10-30 KIOPS [9,10]. Furthermore, these drives consume 0.2W while idle and 2W at full speed [11]. Using these disks, we take the current trend of dividing data into multiple partitions across multiple servers [5] to its logical extreme: use a separate CPU and host for each disk, building the cyber-brick that Jim Gray originally advocated [2]. In fact,

if we pair an SSD with one of the recent energy-efficient CPUs used in laptops and netbooks (e.g., Intel's Atom N270 [8] clocked at 1.6GHz), we arrive at an Amdahl number close to one. Moreover, the IOPS Amdahl ratio is very close to ideal: a 1.6GHz CPU would be perfectly balanced with 32,000 IOPS, close to what current SSDs can offer. Given its balanced performance across all the dimensions mentioned in Amdahl's laws, we term such a server an *Amdahl blade*. Adding a dual-core CPU and a second SSD to such a blade increases packing density at a modest increase in power since the SSDs consume negligible power compared to the motherboard.

5.2.2.1 System Components

To build our cluster, we chose the Zotac ION motherboard as the basis of each cluster node. This motherboard has an Intel Atom N330 dual core CPU and an NVIDIA ION GPU and chipset. This chipset contains 16 GPU cores (heavily multi-threaded SIMD units) and supports 3 SATA drives and 4GB of memory. The ION chip also acts as the overall memory controller for the system, with the GPUs and the Atom processor sharing memory space. Using such systems, we built a 36 node cluster (figure 5.1). The nodes are connected to a single Dell 48-port 1Gb/s Ethernet switch. The nodes are organized into four rows of nine nodes, each with different disk configurations. The aggregate parameters of the cluster are as follows:

- 72 CPU cores + 576 GPU cores
- 144GB total memory

- 36TB hard disk space
- 7.56TB of SSD space
- 18GBytes/s aggregate sequential I/O
- 1,200W of power consumption
- Amdahl Number of 1

5.2.2.2 Cluster Comparison

Table 5.1 compares specs between the GrayWulf nodes and the Amdahl nodes.

System	CPU (GHz)	SeqIO (GB/s)	RandIO (kIOPS)	Disk (TB)	Power (W)	Amdahl Numbers		
						Amdahl Number	Memory Ratio	IOPS Ratio
GrayWulf	21.3	1.5	6.0	22.5	1150	0.56	1.13	0.014
Amdahl	3.2	0.5	10.4	0.50	30	1.25	1.25	0.163

Table 5.1 Performance and power characteristics of the systems compared.

5.2.2.3 Data and Storage Layout

Our experiments focus on maximizing the aggregate sequential I/O performance. True to our scale-down-and-out spirit, the basic building blocks consist of a single low power Mini-ITX motherboard with 3 disk drives. Initial I/O experiments showed

Figure 5.1 The assembled 36-node cluster.

that using the dual Atom Zotac boards with their three internal SATA channels leads to a solid 500MB/s sequential read performance using two high performance SSDs, with write speeds also reaching 400MB/s. The only disadvantage of these systems is that given SSD prices we could not afford to buy larger than 120GB disks – an issue which we expect market forces to rectify over time. In order to balance the smaller amount of SSD storage, we use a number of hybrid nodes in which one SATA port

still contains an OCZ Vertex II 120GB drive, while the other two ports have a mix of an OCZ, a Samsung Spinpoint F1 1TB 3.5in drive (128MB/s at 7.5W), or a Samsung Spinpoint M1 0.5TB 2.5in drive (75MB/s at 2.5W). While the Samsung drives have lower sequential I/O performance compared to the SSDs, we can still almost saturate the motherboards throughput and at the same time attach a lot more disk space. We note that the second and third SATA ports share a port expander, thus their net aggregate throughput is limited to less than 250MB/s. Table 5.2 summarizes the cluster disk configuration.

Node Row	SSD Count	1TB Count	0.5TB Count
1	3	0	0
2	2	1	0
3	1	2	0
4	1	0	2
Total	63	27	18
TB	7.56	27	9

Table 5.2 Per-node disk configurations for the four rows of the Amdahl cluster, and aggregate quantities.

The total data used for the tests consists of three copies of a 2.4TB database, derived from the SDSS DR7 archive [1], for a total volume of 7.2TB. The 544M rows of astronomy data (see later) are partitioned into 36 equal size slices, each of these residing on 3 different servers for redundancy (and added performance). According to this partition strategy, each blade $N \in \{1, \ldots, 36\}$ hosts slice N on its first disk,

slice $(N+1)\,mod\,36$ on its second disk and slice $(N+2)\,mod\,36$ on its third disk. For each slice an additional 12 way partitioning is pre-computed on the head node (but the data is not physically split). This extra 12-way split is represented as ranges of the primary keys in the tables, enabling easy dynamic load balancing depending on the performance of the individual disk volumes. Each job can select a certain number of partitions on different servers and volumes for the shortest lapse time. Figure 5.2 illustrates the partition layout across the cluster.

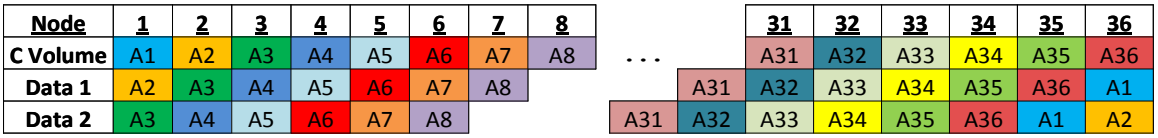


Figure 5.2 Illustration of the data slicing and replication. Each data slice is stored on three of the nodes, shifted by one each time. This enables a dynamic load balancing and a 3-way fault tolerance..

5.2.2.4 Software Configuration

The operating system on the cluster is Windows 7 Release Candidate. The database engine is SQL Server 2008. The installation of these components is fully automated across the cluster. For data partitioning and workflow execution we deploy our own middleware, originally written for the GrayWulf project. The statistical analysis is done in our own implementation of the Random Forest algorithm, written in C (for CUDA) and C# .NET for Windows. We have written a Random Forest implementation in C (for CUDA) that interfaces directly with the database. This

turned out to be not as easy as it first seemed - we had to jump through many hurdles before we could integrate SQL Server and CUDA, and have the CUDA drivers run in a monitor-less configuration.

5.3 The Random Forest Implementation

5.3.1 Predicting on New Data

In the simplest case wherein we seek to produce a regression tree prediction for a new input object in a single processor thread, the naïve implementation is just to follow the decision nodes down the tree using the input values of the new object. This is of course O(h) where h is the height of the regression tree, which is entirely dependent on the training data and the outcome of those random decisions made while growing the tree. But with an eye toward addressing future data volumes like the Panoramic Survey Telescope & Rapid Response System (Pan-STARRS) [1] , which will produce data at rates higher than 1 TB per night, and the Large Synoptic Survey Telescope (LSST) [2] , which will produce approximately 30 TB per night, our goal is to leverage the massive parallelism offered by General Purpose GPU (GPGPU) computing, specifically using NVIDIA's CUDA platform [3] (figure 5.3). This requires the development of a Random Forest regression package in CUDA, and the integration

[1] http://pan-starrs.ifa.hawaii.edu/public/
[2] http://www.lsst.org/lsst
[3] http://www.nvidia.com/object/what_is_cuda_new.html

of such a package into a robust system capable of efficiently feeding the data to the GPU and archiving the results. The Sloan Digital Sky Survey (SDSS) [4] stands as a prototype for these new large surveys, and is thus an ideal environment in which to develop the sort of database-integrated packages that we consider to be the next step in data analysis systems.

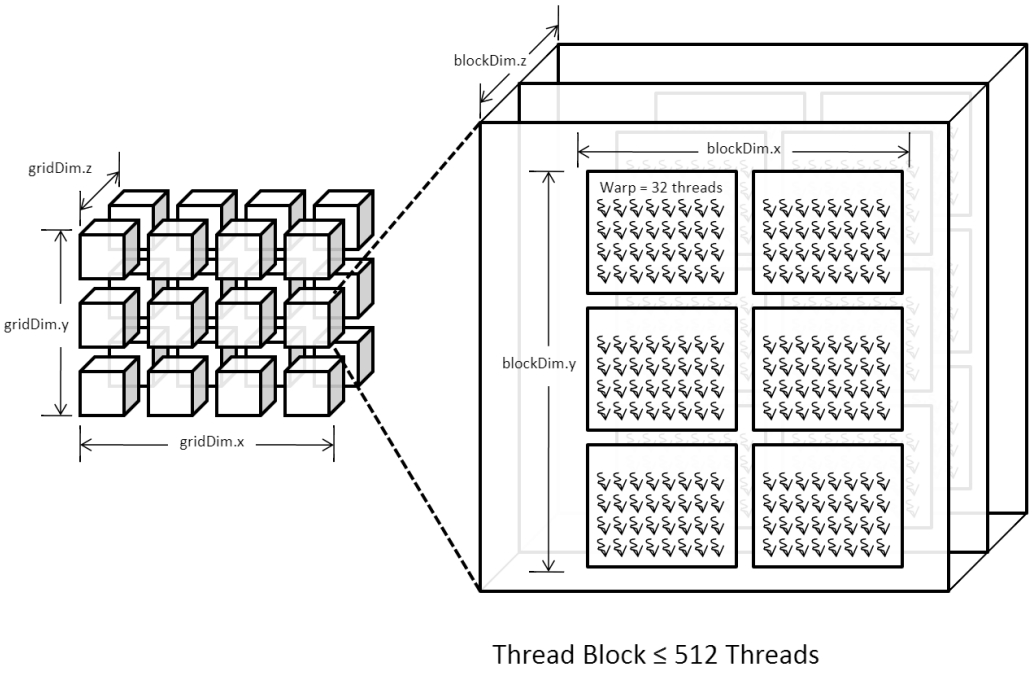


Figure 5.3 CUDA architecture model. The GPU is presented as a three dimensional matrix of threads grouped hierarchically into warps, thread blocks, and a grid. All threads are grouped into warps of 32 threads each. Warp dimensions are undefined and not programmable. Thread blocks may contain up to 512 threads each, and the dimensions of a block are programmable, measured in threads (not measured in warps). Grid dimensions are also programmable, and measured in thread blocks. Maximum number of threads available are on the order of tens of thousands for a typical CUDA-capable GPU.

[4] http://www.sdss.org/

As one can see from figure 5.3, the CUDA architecture is particularly well-suited to data-parallel computation in up to three dimensions, though higher dimensional data and other parallel programming patterns are of course achievable. One crucial constraint on algorithms designed for implementation in CUDA is that the architecture utilizes what NVIDIA calls Single Instruction Multiple Thread (SIMT) execution, wherein all executing threads in a warp execute the same instruction at the same time. This has several consequences which we will address in the description of our current regression algorithm. Not pictured in figure 5.3 are the additional constraints imposed by memory architecture. CUDA exposes to the developer at least three different types of memory (global, shared, and local), each with its own strengths, weaknesses, and constraints, as well as an additional interface to global memory via texture references which can accelerate substantially certain types of memory accesses. Global memory is by far the most plentiful, but also among the slowest, and parallel accesses to global memory become serialized if one violates certain coalesced access patterns. Accesses to shared memory are significantly faster than to global memory, but shared memory is scarce (16kB per thread block) and it must also be accessed in fairly arcane patterns in order to perform optimally. Local memory is small and, ironically, slow. NVIDIA claims that texture memory offers accelerated reads from global memory in cases where proximal threads access proximal texture memory, with the additional benefit of not requiring coalescing. We have not ex-

perimented to determine what constitutes "closeness" among threads and sections of memory, but in informal testing we did observe a speedup of approximately 4x in our algorithm using texture references versus direct global memory reads on a GeForce 9400 video card.

With these intuitive architectural notions in mind, and with the goal of developing a complete database-GPU-database round-trip system in time for the Supercomputing 2009 conference, we settled on the following algorithm design. Our problem as currently defined is intrinsically fully data-parallel – the regression computation for one new data object is completely independent of the results of every other new object, so we use a single thread to compute a single regression estimate for a single new object, and we specify the "geometry" of the execution to be a one dimensional array of threads. Since a final regression estimate for a given object is the average of the individual regression tree estimates for that object, we accumulate the average of the individual tree estimates as we iterate through trees. So the general outline is:

- Load regression trees into global memory
- Load new input data into global memory
- For each tree
  - Bind textures to the current tree in global memory

- Synchronously run new data objects through the current tree naïvely, with adjacent threads operating on memory-adjacent objects
- Store the cumulative results for each new data object

To minimize the sparseness of memory accesses, we encode our regression trees breadth-first. The naturally recursive descent through the regression trees is unrolled into an iteration. Thus, as a consequence of the SIMT execution model, all new data items descend through the tree at the same rate, and are at the same level of the tree at the same time. Since the trees are encoded breadth-first, texture fetches are then limited to a small, contiguous section of underlying global memory, which should, in turn, result in faster texture fetches than if the trees were encoded depth-first. One other consequence of this algorithm design is that one can clearly see that the total time to complete the regression on any given new object is equal to the *worst case* among all new objects assigned to threads in the same warp; objects whose leaf nodes reside higher in the tree reach their stopping criterion (reaching a leaf node) before objects whose leaf nodes reside in the lowest level of the tree, but a thread which completes early can't take another job until all other threads in its warp complete. One common pattern in CUDA programming is to have threads "walk" through memory with a stride equal to the "width" of the execution configuration in threads. But in informal testing, this resulted in no performance gains, presumably because it

results in sparse global memory accesses which become serialized.

As a prototype, our CUDA Random Forest implementation performed very much in line with our expectations, as described in the following section. But two particular use cases offer intuition on a possible faster approach to traversing the regression trees. In the simplest case of one-dimensional real input data, figure 5.4 suggests an approach based on binary search. This reduces the worst case running time on the worst possible tree (a maximally unbalanced tree) from O(n) to O(lg n). Since SIMT execution effectively makes the average case running time equivalent to the worst case within a warp, reducing the worst case running time should result in a significant gain overall. Capitalizing on this one-dimensional intuition will require generalizing to higher dimensions and accounting for the additional computational and implementational complexity induced by the randomization process.

5.3.2 Integrating with SQL Server

Some time ago we had initially developed a prototype demonstrating a novel way of integrating CUDA with SQL Server seamlessly by wrapping C/CUDA functions and data structures in the Microsoft Common Language Runtime (CLR) using a

Figure 5.4 Two possible regression tree morphologies. Note that in each of these two cases, for one-dimensional real input data, the leaf nodes represent disjunct intervals on the real line, and they will be in order. Thus prediction need not be implemented as a top down traversal, but may be recast as a search among leaf nodes. With the constraint of ordered leaf nodes, as holds in these two cases, the search may be implemented as an O(lg n) binary search. (a) A complete tree. Prediction time is O(lg n) for both naïve traversal and binary search among leaves. (b) A maximally unbalanced tree. Prediction time is O(n) for naïve traversal, but O(lg n) for binary search among leaves.

marshaling framework called *P/Invoke* and the CLR facilities available in SQL Server (SQLCLR). We then sought to expand this prototype to accommodate more powerful general purpose data analysis within SQL Server without straining CPU resources, and the first use case to this end became the project in which we integrated our CUDA Random Forest package with SQL Server. Our initial prototype had been developed on Windows XP. We discovered that as a result of driver issues which we can't go into due to a "gentleman's NDA", the GPU driver was no longer available to SQL Server in Windows versions starting with Vista. NVIDIA is addressing this issue, but details on that are also subject to NDA. There is, however, an additional limitation in the SQLCLR which prevents CLR code from spawning threads. This constraint prevents parallel execution of input and output streams for retrieving input data and writing results to the database in stream-like fashion, so very little useful work can be done with the GPU in this setting. Development of multi-threading support in

the SQLCLR appears not to be a priority for the SQL Server development group, so this constraint is likely to remain for the foreseeable future. In the meantime we resolved to achieve the desired behavior, the ability to drive GPU-enabled data analysis from within SQL Server, utilizing a more traditional software architecture in which an outside user process (the "GPU server") polls a job queue table within SQL Server. The job queue contains, among other necessary information, columns for "input query", a "job type" (i.e. Random Forest regression using the GPU, regression using the CPU, perform a NOOP, etc...), "job status", and a "destination table", which in its simplest form is all the GPU server requires to be able to access the desired data set for which to produce regression predictions and to write the results back to the database using the database interop facilities in Microsoft's .NET framework. So the execution flow in our database-GPU-database analysis system is as follows:

1. User adds entry to job queue table in SQL Server using SQL DML. The entry includes an input query expressed in SQL DML and a results table name. The table is predefined with columns for ID, regression estimate, and error distribution sigma.

2. The GPU server polls the job queue table and accepts the job.

3. The GPU server opens a query stream to the database using the input query specified by the job request. Another database connection is opened for the

purposes of writing the results back to the database.

4. The GPU server iterates through batches of input data.

   (a) Input data is read into a buffer, then copied into global memory on the GPU.

   (b) The GPU server executes the prediction GPU code.

   (c) The GPU server writes the results to the output stream.

5. When GPU processing is finished and the results are fully written to the destination table, the GPU server updates the job queue table with a status indicating completion.

5.4 The Experiment

5.4.1 Photometric Redshift Estimation on the Amdahl Cluster

To test this new system, we use the photometric redshift estimation problem described in chapters 2 and 3. In our experiments we use data from the Sloan Digital Sky Survey DR7 archive [1]. The original data is in the form of a 5TB SQL Server database. We partitioned and replicated 2.4TB across the nodes for parallelism and

faster access (see Figure 5.2). The galaxies are selected dynamically with a SQL query from the PhotoObjAll table, with a cardinality of 544M.

We only use galaxies with a good detection in all five of the SDSS photometric bands. A subsample of the galaxies is also derived from using the different flags describing the quality of the image segmentation, and a flux limited sample is also used. We compute the Random Forest (RF) estimator [14] to their photometric redshift [15,16] and write the results to disk, for about 128M objects. The estimator is using a combination of the flux measurements and a measure of the inclination of the galaxy. The code performs the RF estimation in two ways: one utilizes just the Atom processors, while the other makes full use of the GPUs. We also estimate the pure I/O need of the test using a NULL test, in which we read the input data, put zero as the estimator and then write the output to disk. We performed the test on all nodes and on all disk volumes. A few aggregate estimators are computed on the result set, assessing the quality of the result. On all the nodes we experiment with selecting a vertically partitioned table of the clean galaxies, saving it on the SSD disks, and running the rest of the analysis from there. A multi-step workflow is run, which first selects and filters the data and then applies the statistical regression coefficients for each tree on each object (approximately $30x300M = 10^{10}$ tree searches and regressions). Finally we use a subset of the result set for which we have testing

data available, and evaluate the accuracy of the algorithm as a function of the distance of the galaxy. The problem of statistical redshift estimation is becoming critical, since in the next generation of sky surveys (LSST, Pan-STARRS in less than four months) we will have several billions of galaxies with photometric measurements, but no direct distance. At that point this approach will be the only way to obtain cosmological distances, and a fast, scalable algorithm is of utmost importance. In the next paragraphs we discuss the details of the test codes. There are three different versions of the code, one that uses the GPU, one that uses the Atom CPU and one that only does I/O operations to see the I/O component in the execution times.

Common

We use three test implementations: GPU regression, CPU regression, and NOOP regression, hereafter referred to as GPU, CPU, and NOOP for the duration of this section. All I/O is implemented using the .NET IDataReader interface for input to the regression function and IDataReader also for output directly into SQL Server via SqlBulkCopy. The GPU and CPU implementations test regression tree traversal performance on the GPU and CPU, respectively, while the NOOP implementation reads input data just as the GPU and CPU implementations and outputs zero values instead of doing any actual regression. This is intended to measure the time spent simply doing I/O.

GPU

Since streaming data to the GPU would be inefficient even to the extent that its technically possible, we run GPU regression in batches of 50,000 test objects at a time. The GPU regression function is wrapped in an IDataReader implementation which is fed to SqlBulkCopy. When SqlBulkCopy first attempts to read from the GPU IDataReader, this triggers the spawning of a separate host (CPU) thread which initiates sequential batch executions on the GPU. Each batch result is placed in an output queue from which the GPU IDataReader (running in the main host thread) pulls data to return to SqlBulkCopy. Thus processing is done in bursts, and output data tends to be written in corresponding bursts. This intermittent loading is also reflected in power consumption; when running the GPU regression, power consumption escalates to some baseline level with peaks significantly above that level on GPU processing bursts. In the future we will likely tailor the I/O rate to match the rates seen in video gaming, the application for which the GPUs were designed.

CPU and NOOP

These implementations are easily streamed. The CPU regression is wrapped in an IDataReader implementation which is fed to SqlBulkCopy. SqlBulkCopy requests a row from the CPU IDataReader, which in turn requests a row from its input IDataReader and regresses on that data. The regression is implemented in C# and is identical to that of a single thread in the GPU implementation, using the same

binary encoded regression trees and the same traversal algorithm. In fact the GPU is at a disadvantage here as each test object in the GPU takes as long as the slowest object in the same thread block. This GPU implementation is suboptimal, but it is simple to code, understand, and maintain, and is still very fast in practice. A more sophisticated implementation may cost as much in planning overhead as this simple implementation loses in efficiency.

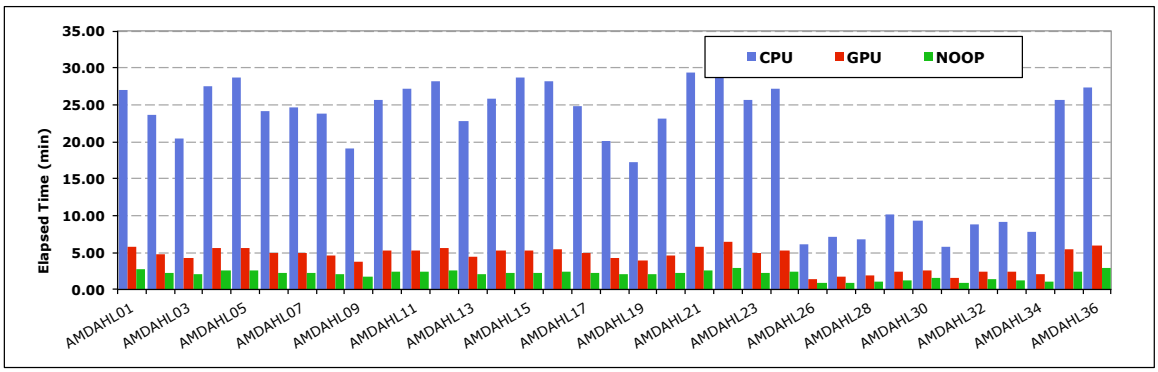


Figure 5.5 The results of the Random Forest tests over the narrow data table indicate that even with the GPUs most of the time is spent in the computation part. It is also clear that the use of the GPUs yields a factor of 4.5 performance gain over using the Atom CPUs, or a factor of 8 if we subtract the I/O time (NOOP).

5.4.2 Results

Using the narrow table, the results indicate that GPU version of the code performs almost a factor of 5 faster than using the Atom processors, even though tree traversal is one of the worst algorithms to run on the GPU, since we lose code coherence very rapidly as each object does its own tree traversal. One can see that all nodes finished

well under 500 seconds we were able to process 128M objects times 50 trees, i.e. 6.4B tree-traversals in less than 5 minutes! The best figure of merit for these tests is $(CPUNOOP)/(GPU - NOOP)$, the ratio of the pure computational budget. The average of this was measured to be 8.08 ± 0.8. The average of CPU/GPU is 4.67 ± 0.6. When compared to a single GrayWulf node on one of the partitions, the GW lapse time was 289 sec, compared to 331 sec on a single Amdahl node using the GPUs, i.e. a single Amdahl node can keep up with a much more powerful server (at least in a single threaded application). The CPU lapse times remain essentially unchanged when using the wide table, while the GPU numbers go up slightly. This indicates that in the CPU test most of the I/O is in the background.

The power consumption

Throughout all the tests the power drawn has varied between 24W per node to 45W per node (with the two hard disks). The aggregate power consumption for the whole cluster was between 885 and 1261W, typically less than a single GrayWulf node.

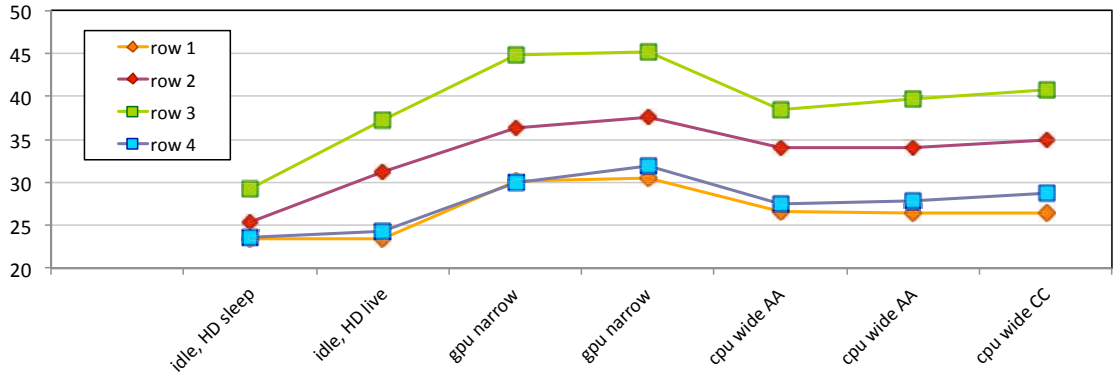


Figure 5.6 The power consumption during our various tests. The most power is drawn when the GPUs are at full load.

5.5 Discussion

We presented a low-power compute cluster which, running our CUDA Random Forest regression implementation on the SDSS photometric redshift dataset, performeed comparably to a single-threaded CPU reference implementation (written by us) running on a single CPU core of a conventional cluster node with two quad-core Intel Xeon 2.66GHz processors, 24GB RAM, and consuming 1150W, approximately the power consumption of the entire low power cluster. Using the more balanced low power cluster allows us to scale I/O bandwidth with drastically lower power consumption than with a conventional cluster, and availability of low-power onboard GPU's allows us to do this all without sacrificing the ability to do meaningful scientific data analysis *in situ*. More detailed results are presented in Szalay et al. [57].

Chapter 6

Analytic Congruence of Random Forest Regression and Kernel Regression

6.1 Background

Since its introduction, Random Forest regression [10, 38] has become a popular method of non-parametric regression. At the time of this writing, a Google Scholar search for the phrase "random forest regression" (quotes included) restricted to 2010 publications and later yields more than 1500 academic articles. Random Forests perform so well and are so convenient that they gain specific mention for their popularity

by Anthony Goldbloom, CEO of data science crowdsourcing website Kaggle[1]. Sadly, the context of this mention is a comment on the displacement of Random Forests by another algorithm. With luck, perhaps our work may assist in restoring Random Forest regression to its rightful place at the top.

6.2 Kernels in Random Forest regression

The first question is, "Why? What's the significance?" There are at least three reasons we might wish Random Forest regression to resemble kernel regression. The first and most important is that though it seems unlikely that Random Forest regression is strongly universally consistent, any analytic resemblance to kernel regression can only be a good thing, since the latter has this desirable property [28, Ch. 10]. Furthermore, at least for functions which can be transformed into densities, kernel regression's optimal rate of convergence cannot be improved on [61, Ch. 6]. Finally, as we will demonstrate, the congruence gives us at least some conceptual insight into how we might improve on the standard Random Forest regression methodology. This then begs the question, "why not just use kernel regression instead?" One answer is that Random Forest regression effectively does much of the distance computation at training time, leaving only a relatively small number of tree traversals rather than n distance comparisons to be executed for each new observation.

[1] http://youtube.com/watch?v=GTs5ZQ6XwUM

6.2.1 Analytical model

We are given a training set $\mathcal{D} = \{(X_i, Y_i) : i \in \{1, \ldots, n\}\}$, drawn from an unknown joint probability distribution F_{XY} with $|\mathcal{D}| = n \in \mathbb{N}$, and for our purposes, we may assume without loss of generality that the (X_i, Y_i) come ordered with the X_i non-decreasing. One may recall from Breiman [9], Breiman et al. [11], Ho [38], and Breiman [10] that Random Forest regression works as follows. We train t Random Forest regression trees, $t \in \mathbb{N}$, on corresponding bootstrap samples $\mathcal{D}_j$ drawn from $\mathcal{D}$, $j \in \{1, \ldots, t\}$, generate individual regression tree estimates on our new data, then for each new observation, and take the mean of the individual tree estimates. These means are our final regression estimates. A given individual Random Forest regression tree is trained by iteratively partitioning the training set using binary axis-parallel splits at each node, choosing the splitting dimension and split point in that dimension from a random subspace of the input space according to minimum sum of weighted empirical risks among the child nodes. This is done recursively, stopping at nodes which would result in child node sizes falling below a prescribed minimum observation count.

Let us consider a test point x_0 and try to estimate the conditional mean $\mathrm{E}[Y|X = x_0]$ using a Random Forest denoted $\mathcal{F}$. We know that $\mathrm{P}((X_i, Y_i) \in \mathcal{D}_j) < 1$ for all reasonable choices of bootstrap sample size and for all choices of i and j. Furthermore, it will generally be the case that $\mathrm{P}((X_i, Y_i) \in \mathcal{L}_j(x_0)|\ (X_i, Y_i) \in \mathcal{D}_j) < 1$ for particular choices of i and j, where $\mathcal{L}_j(x_0)$ denotes the set of training observations residing in

x_0's leaf node (allowing duplicate elements to accomodate sampling with replacement) in $\mathcal{F}$'s jth regression tree. The product of these probabilities is simply $\mathrm{P}((X_i, Y_i) \in \mathcal{L}_j(x_0))$, and let us call that $p_{t,i}(x_0)$ since the trees are identically distributed and independent conditioned on the training data, and therefore exchangeable. Let

$$\hat{p}_{t,i}(x_0) = \frac{1}{t}\sum_{j=1}^{t} \frac{\#\text{ times } (X_i, Y_i) \text{ occurs in } \mathcal{L}_j(x_0)}{|\mathcal{L}_j(x_0)|}, \tag{6.1}$$

$$\lim_{t\to\infty} \hat{p}_{t,i}(x_0) = p_{t,i}(x_0). \tag{6.2}$$

This seems quite obvious since adding trees is equivalent to extending trials to binomial random variables representing counts of each training observation's contribution to the regression estimate. What, then, is the Random Forest regression estimate $\hat{\mu}_{Y|x_0}$ of $\mathrm{E}[Y|x_0]$ but $\sum \hat{p}_{t,i}(x_0)Y_i$?

Kernels traditionally are defined as non-negative symmetric functions with unity integral. Clearly p as we have defined it, for any choice of $\mathcal{D}$ and x_0, satisfies the non-negativity and unity integral conditions, being as it is an empirical probability distribution function. We should not assume that it will be generally symmetric, though it stands to reason that often it may be approximately so. We may relax the symmetry condition, call p a member of a family of kernels, and benefit from analyzing it as such. The focus of our attention for this investigation is how to apply

the notion of *kernel bandwidth* to Random Forest regression. We have a conceptual model of p as a kernel, and recalling our ordering assumption, we expect that p will decrease monotonically as we move away from x_0 in "ordinal space", and therefore in a probabilistic sense, also in real space since the X_i are drawn according to the X-marginal of F_{XY}. We can imagine p as looking something like a uniform or unimodal histogram centered on x_0 and with bin locations determined by $\mathcal{D}$.

For the sake of simplicity, let us for the moment assume a uniform shaped p centered exactly on x_0 and compare it with Nadaraya-Watson kernel regression with a box kernel which we'll call K. Recall that in such a regression model the weight of the ith observation's response in computing the regression estimate is determined by

$$w_i = \frac{I(\frac{x_0 - x_i}{h})}{\sum_{j=1}^{n} I(\frac{x_0 - x_j}{h})} \tag{6.3}$$

where $I(x) = 1$ for $|x| < 1$, 0 otherwise [46, 62]. Scaling the distances of training observations x_i from x_0 by $\frac{1}{h}$ draws more observations into $K's$ support for larger choices of h, effectively distributing K's mass over a wider range of training observations. How do we achieve the same effect with our Random Forest kernel, p? Observe that membership in p's support is indicated for a training observation x_i directly by the existence of a non-zero value of $\hat{p}_{t,i}(x_0)$, and let us denote $|\{i : \hat{p}_{t,i}(x_0) > 0\}|$ as

N_+. Now let us consider the following statistic:

$$W = \frac{1}{\sum p_{t,i}(x_0)^2}, \tag{6.4}$$

the inverse sum of squared probabilities of contributing to the regression estimate of x_0. Since we are, for the moment, concerned only with uniform kernels, the value of W is invariant to the number of times any contributing observation contributes, so that W comes out equivalent to N_+. In fact we observed something very close to this in anecdotal trials with a forthcoming Random Forest implementation which outputs both of these numbers for us, using the empirical plugin estimates.

6.2.2 t, h, and their Selection

One potential beneficial by-product of the model we propose is some analytical guidance on selection of the forest size parameter t. How many trees are sufficient? That depends on how many distinct observations we must accommodate, and the rate at which we accumulate them as we add trees to our forest. Let us therefore attempt an estimate of N_+, which is determined by b.

Given a point of interest x_0 and a bootstrap sample size b, what can we expect N_+ to look like? It seems reasonable for us to conjecture that the expected number

of distinct observations in a leaf node is invariant to choice of x_0. The support of the distribution of this node cardinality will be $\{m, \ldots, 2m-1\}$, or $\{5, \ldots, 9\}$ for a typical choice of $m = 5$, where m denotes our chosen minimum node size. It is therefore reasonable to conjecture that $E[M] \approx 7$, where M is a random variable denoting the node cardinality. In one dimension, then, we expect a node to cover a range in X described by the outer two of seven sequential order statistics centered on x_0. Let us then consider ordering $\mathcal{D} \cup (x_0, E[Y|x_0])$ by x, and suppose that j is x_0's expected position in this ordering over arbitrarily many realizations of $\mathcal{D}$. Then the range we seek is expressed by

$$E[X_{((j+3):b)}] - E[X_{((j-3):b)}] \tag{6.5}$$

where $X_{(r:b)}$ denotes the rth order statistic from a sample of size b drawn from X's distribution. In fact this is a slight oversimplification; we will actually be subsampling b observations from a sample of size N, but let's not clutter our exposition with this detail. It's easy enough to correct by repeated application of the same idea.

Consider the synthetic joint distribution pictured in Figure 6.1. In our distribution, $X \sim U(-4, 4)$ continuous, and $Y|X \sim N(\mu_{Y|X}, \sigma_{Y|X})$ where $\mu_{Y|x}$ is a damped oscillating function with varying frequency, and $\sigma^2_{Y|x}$ is a linear sweep from 0.01 to 0.05. In our case, where the X_i are uniformly distributed, we will lose little generality

in our current task by assuming that $j = 4$, so that the expected range of the containing leaf node is described by $E[X_{(7:b)}] - E[X_{(1:b)}]$. The trivial closed-form expression $\frac{r}{b+1}$ for the expectation of the rth order statistic of a sample of b $U(0,1)$-distributed random variables is given by David and Nagaraja [25, pg. 35]. We may simply scale the result to apply to our $U(-4,4)$-distributed random variables, arriving at

$$E[H] = 8 \times \frac{7-1}{b+1} = \frac{48}{b+1}, \tag{6.6}$$

where H denotes the bandwidth of our Random Forest kernel. Now we may reasonably claim that in our special case,

$$E[N_+] \approx \frac{E[H]}{r_X} \times N = \frac{6N}{b+1}, \tag{6.7}$$

where r_X denotes the range of X's support.

Given an estimate of $E[N_+]$, how many trees t must we train to ensure that each contributing x_i is duly represented? We recall our assumption of an approximately uniform-distributed kernel, which gives us that in a given tree, *all contributing observations are equally likely to appear in* x_0*'s leaf.* It's easy to see that this probability is $\frac{E[M]}{N_+}$. Modelling directly how many trees we must train to encounter each contributing x_i in $\mathcal{L}_{z_j}(x_0)$ for at least one choice of $j < t$ seems unpleasant. An easier way to

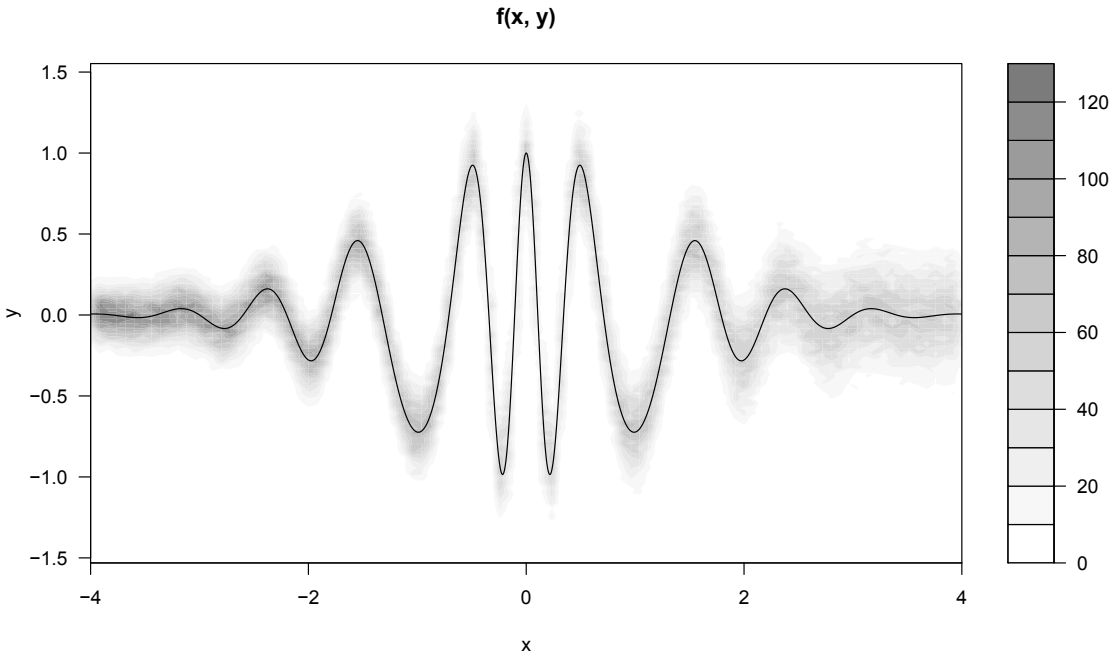


Figure 6.1 Joint density of experimental dataset.

get at t seems as follows.

Let Z_i be a random variable representing the number of trees we must train to encounter our first observation of a particular choice of x_i which is in $\mathcal{L}_{z_i}(x_0)$. We can see that $Z_i \sim Geom(\frac{E[M]}{N_+})$. Now let's assume that the Z_i are only weakly dependent; then we may consider the distribution of $Z_{(E[N_+])}$, the tree count required to observe the most reluctant of our $E[N_+]$ kernel participants. Then we may attempt to impose an upper bound on $E[Z_{(E[N_+])}]$, and in fact this is given to us by David and Nagaraja [25, pg. 62]. Applying it to our special case gives us $E[Z_{(E[N_+])}] \leq \mu_{Z_i} + \frac{(E[N_+]-1)\sigma_{Z_i}}{(2E[N_+]-1)^{\frac{1}{2}}}$,

where μ_{Z_i} and σ_{Z_i} are given to us by the Geometric distribution. In our case this is $O((\frac{N}{b})^{\frac{3}{2}})$, which should give us an approximate practical upper bound on t.

None of this helps us in the general case. The natural first instinct when contemplating automation of t's selection is to base our Z_+ estimate on local histogram approximations of the joint distribution, but of course this is the sort of labor we're trying to avoid with Random Forest regression. Since the expected marginal gain in contributing observations decreases monotonically with each tree, it would probably be just as well to add trees as long as the gain in N_+ for at least one OOB training observation exceeds some sensible threshold like 1.

6.3 Looking at MSE by Sample Rate

The point of all this, of course, is to improve the performance of Random Forest regression as we apply it. To that end we will investigate the relationship between b and the mean squared error of our regression estimates as a function of a one-dimensional observable using the synthetic distribution described in the previous section (Figure 6.1). In particular, we're interested in any potential benefit from conditioning our choice of b on the value of X_i. We want to see how this relationship plays out near extrema of the response function, in regions with more stable trends, and in regions of higher and lower conditional variance.

Consider, for instance, if we set our point of interest $x_0 = 0$. How do we expect the MSE of $\hat{\mu}_{Y|0}$, taken over many trials, to respond to different choices of b? The minima of the μ function about $x_0 = 0$ occur at approximately $x = \pm 0.22$, and by equation 6.6 we expect setting $b = 100$ to bracket these points somewhat closely. We shall see that this does very poorly on $\hat{\mu}_{Y|0}$ in accordance with our intuition that using such a kernel on that point is the very worst thing we can do with this data.

In fact this portion of the μ function can be approximated conceptually by two line segments symmetric about 0. Then we can ponder the bias-variance decomposition of the MSE of $\hat{\mu}_{Y|0}$ somewhat conveniently. For any uniform kernel narrower than this width, the magnitude of the bias would be approximately the distance to the midpoint of the portion of one of the line segments falling within the kernel. Since $\mu_{Y|x}$ ranges over $[-1, 1]$ as x ranges over approximately $[-0.22, 0]$, the slope of this line segment is approximately 9, so that with equation 6.6 we get

$$Bias(\hat{\mu}_{Y|0,b})^2 \approx \left(\frac{1}{2} \times \frac{1}{2} \times 9 \times \frac{48}{b+1}\right)^2 = \frac{11664}{(b+1)^2} \tag{6.8}$$

Since we assume a uniform kernel, $\hat{\mu}_{Y|0}$ is essentially a local sample mean, and as such, submits to the Central Limit Theorem. Consequently $Var(\hat{\mu}_{Y|0}) \approx \frac{Var(Y||X-x_0|<h)}{N_+}$.

The Law of Total Variance gives us the numerator; it is the mean of the conditional variance of Y within the window (equivalent to $Var(Y|0) = 0.03$ in our case) plus the variance of the conditional mean of Y again restricted to the window. Since $\mu_{Y|0}$ is approximately uniformly distributed on the aforementioned line segment we end up with

$$Var(Y||X - x_0| < h) \approx 0.03 + \frac{1}{12}\left(\frac{1}{2} \times 9 \times \frac{48}{b+1}\right)^2 \tag{6.9}$$

and with equation 6.7 we get

$$Var(\hat{\mu}_{Y|0}) \approx \frac{b}{6N}\left[0.03 + \frac{3888}{(b+1)^2}\right] \tag{6.10}$$

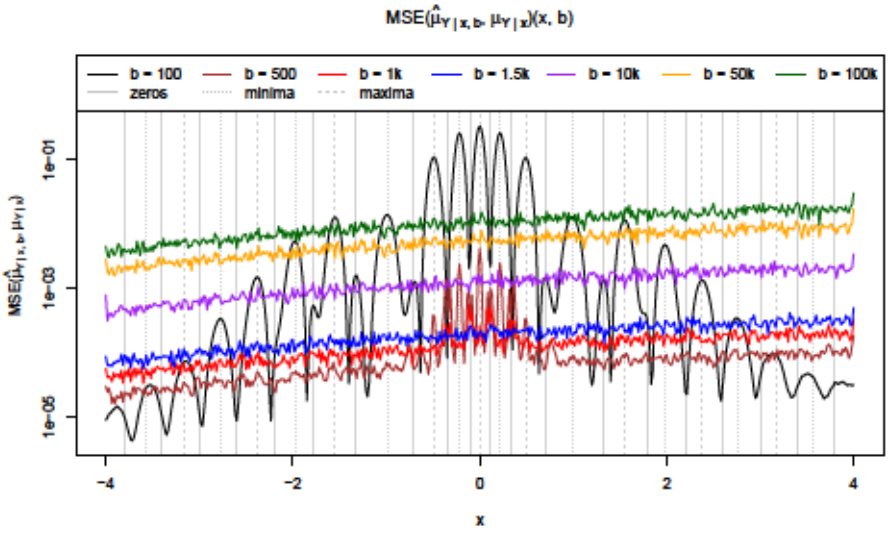
The MSE of the Random Forest regression estimates evaluated at $x_0 = 0$, as a function of b, should then be the sum of the squared bias and variance from equations 6.8 and 6.10.

To test the proposed model and the MSE prediction we have just made, and to visualize how different choices of b affect the MSE of Random Forest regression estimates of the conditional mean over the entire joint distribution shown in figure 6.1, we generate a training set of $100k$ observations and a test set of 480 points evenly spaced over the range of X (with $Y = \mu_{Y|X}$). We then run 100 trials each at bootstrap sample sizes of 100, 500, $1k$, $1.5k$, $10k$, $50k$, and $100k$, with corresponding forest sizes

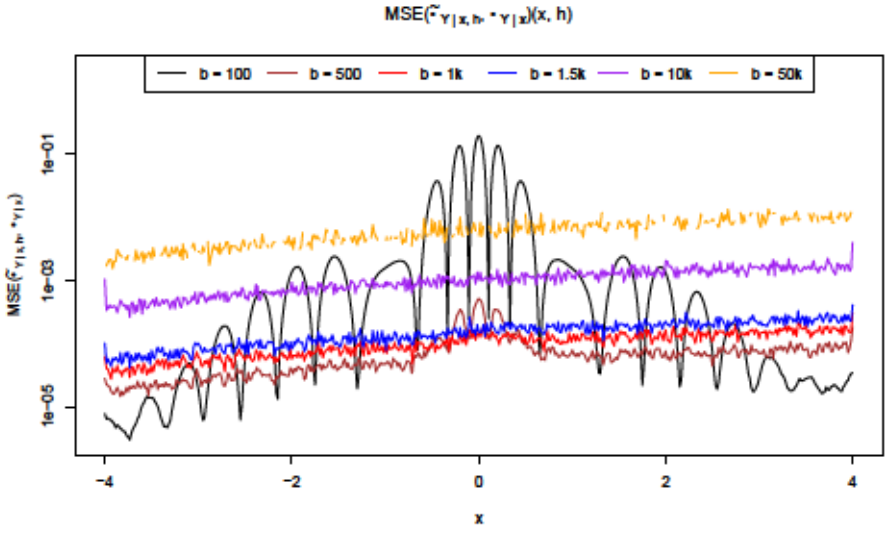
of $10k$, $2k$, $1k$, 500, and 100 trees for the remaining sample rates. We then measure the MSE of the regression estimates as a function of x over the 100 trials at each bandwidth (Figure 6.2(a)).

A few things immediately are clear from figure 6.2(a). First is that the de facto standard bootstrap sample size of $100k$ in this case appears to be almost the worst possible choice! Next is that, of the sample sizes we tested, for most of the range of the distribution we would be best choosing $b = 500$, giving us $N_+ \approx 1200$ resulting in $h \approx 0.096$ versus the narrowest choice of $h \approx 0.00048$ at $b = 100k$. Lastly, at $x_0 = 0$, as expected, the broadest kernels are penalized, and we would prefer $b = 1500$ from among these sample sizes, yielding $h \approx 0.032$.

In the course of producing figure 6.2(a) it seems a convenient time to test how seriously we take our use of the word "bandwidth"; if we believe the congruence which we claim, then let us test our model by comparing Nadaraya-Watson kernel regression at such bandwidths as we claim correspond to the Random Forest bootstrap sample size by way of equation 6.6. We must, however, compensate for the fact that equation 6.6 expresses the *entire* width of our kernel, while conventional kernel regression bandwidth expresses the interquartile range (IQR) of the kernel. For our simplified



(a)



(b)

Figure 6.2 (a) Log MSE of RF regression estimates as a function of x by bootstrap sample size b. (b) Log MSE of Nadaraya-Watson kernel regression estimates as a function of x using bandwidths determined by model $h(b)$. The correspondence is striking over most of the range, though the model breaks down as the sample size approaches N, resulting in bandwidths which are unusably narrow for kernel regression.

model using uniform kernels this is easy; the IQR is simply half the full width of the kernel. We make such correction and measure the MSE of kernel regression as a function of x using a uniform kernel at the bandwidths of interest. We perform 100 trials using the same training data sets used to produce 6.2(a) and plot the MSE taken over those trials versus x in figure 6.2(b). The agreement with figure 6.2(a) is persuasive. However, the model could use some refinement, as it yields overly narrow kernels as the bootstrap sample size approaches N; already at $b = 50k$ there are no training observations within the corresponding classical kernel for many points in the support. By the time $b = N$ the resulting bandwidth is unusable by kernel regression.

In figure 6.3 we plot, for $x_0 = 0$, the MSE we predict for b in our trial sample sizes compared with what we observed in our experiment, substituting $b = 125$ for our prediction to compare with the empirical $b = 100$ simply because we already know the simplified model resulting in equations 6.8 and 6.10 breaks down for $b < 125$. Figure 6.3 is generally consistent with our model, but with an unfortunate and pronounced gap at the low end of the sample sizes. We do not yet have an explanation for this, but suggest that the uniform kernel assumption is not strictly true, and that it may be somewhat dependent on choice of b and the underlying distribution of the data. However, over all we are encouraged by the general agreement with the model. Some refinement is in order.

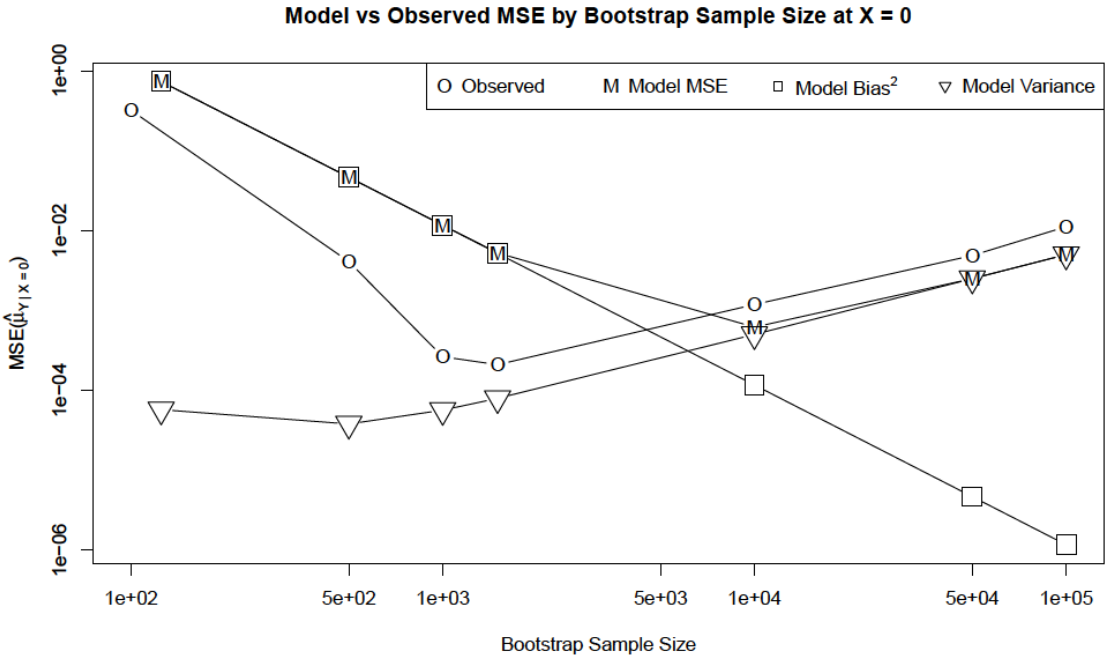


Figure 6.3 Model versus empirical MSE at x = 0.

6.3.1 Estimating MSE Judiciously

In practice we will have neither prior knowledge of the underlying data distribution nor enough data to produce a plot like figure 6.2(a). It seems clear that if time complexity were not a concern, we would learn the sample size as a function of x. For now we will aim simply for a decision rule over two choices of b at each x. It is easy enough to estimate the unconditional MSE with cross-validation, but we will typically only get one observation at each x we observe, so estimating the conditional MSE is not so straightforward. But let us consider the expected bias-variance decomposition of the MSE as shown in figure 6.3. Disregarding, for the moment, that it suggests that the model needs work in the $b \in [500, \ldots, 1500]$ range, it shows that the model

suggests that in this range of bandwidths, the MSE is composed almost entirely of bias. This is, ostensibly, bad news since it's not obvious how to estimate bias locally with limited data for cross-validation. But here the Law of Total Variance comes in handy again:

$$Var(Y|X \in \mathcal{W}) = E_{X \in \mathcal{W}}[Var(Y|X)] + Var_{X \in \mathcal{W}}(E[Y|X]), \tag{6.11}$$

where $\mathcal{W} = [x_0 - \frac{h}{2}, x_0 + \frac{h}{2}]$. The square of the bias is the squared expected distance of Y from $\mu_{Y|x_0}$ in the kernel window while the second component of the RHS of equation 6.11 is the variance of the conditional mean in that same window. These expressions should always move in the same direction as functions of x, while the first component in the RHS of equation 6.11 will remain relatively constant over reasonable size windows in our data set, as well as for many other real world datasets, which often tend not to be wildly heteroskedastic. This is fortuitous, because we have a cheap way to estimate conditional variance as described in Carliles et al. [16]. We can simply estimate and compare the conditional estimator variances at the different bandwidths, and any place the bias would become excessive for a given bandwidth, we should be able to detect this in the estimated conditional estimator variance for that bandwidth.

6.3.2 Experiment

For each of the two choices of b we train a Random Forest on one training set, compute the variance of the individual tree estimates at each choice of x from a second training set, and take those as estimates of $Var(Y|X \in \mathcal{W}(x,b))$ for each x in the second training set. We then train two more forests, one for each choice of b on the x's from the second training set combined with their associated estimator variance estimates to get regression estimators for $Var(Y|X \in \mathcal{W}(x,b))$ (figure 6.4). Then for all x in a third set we produce estimates of $E[Y|x]$ for each choice of b using the original two regression forests. We then run the third x's through the *conditional variance forests* and compute deltas between conditional variances for corresponding choices of x; for deltas exceeding the 90th percentile of the delta distribution (chosen to minimize MSE on this set), we use the original regression estimate for the choice of b associated with the lower conditional variance estimate. Figure 6.5 illustrates. The MSE on this third set is 0.00013 for $b = 500$, 0.00019 for $b = 1500$, and 0.000085 for the hybrid estimator. We should of course test this on a held-out set since we used this third set to determine the b-selection criterion, but the results will look the same, and this particular classifier is not really the point here; a much better one can be constructed with this model in mind.

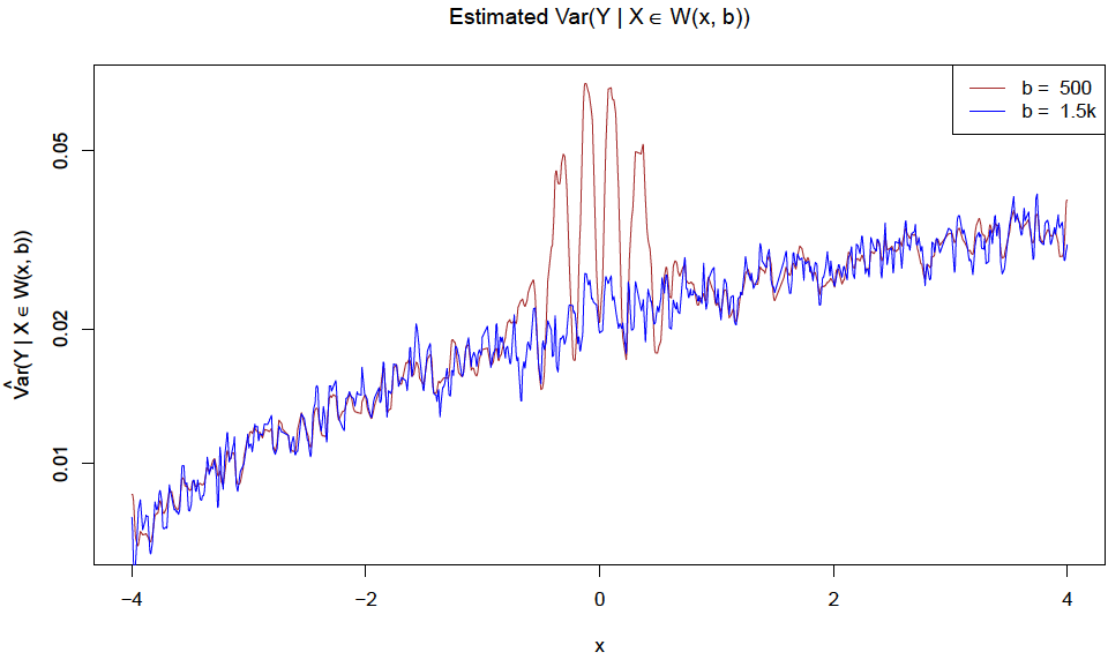


Figure 6.4 Estimated $Var(Y|X \in \mathcal{W}(x,b))$ for bootstrap sample sizes 500 and 1500 using the methodology of Carliles et al. [16].

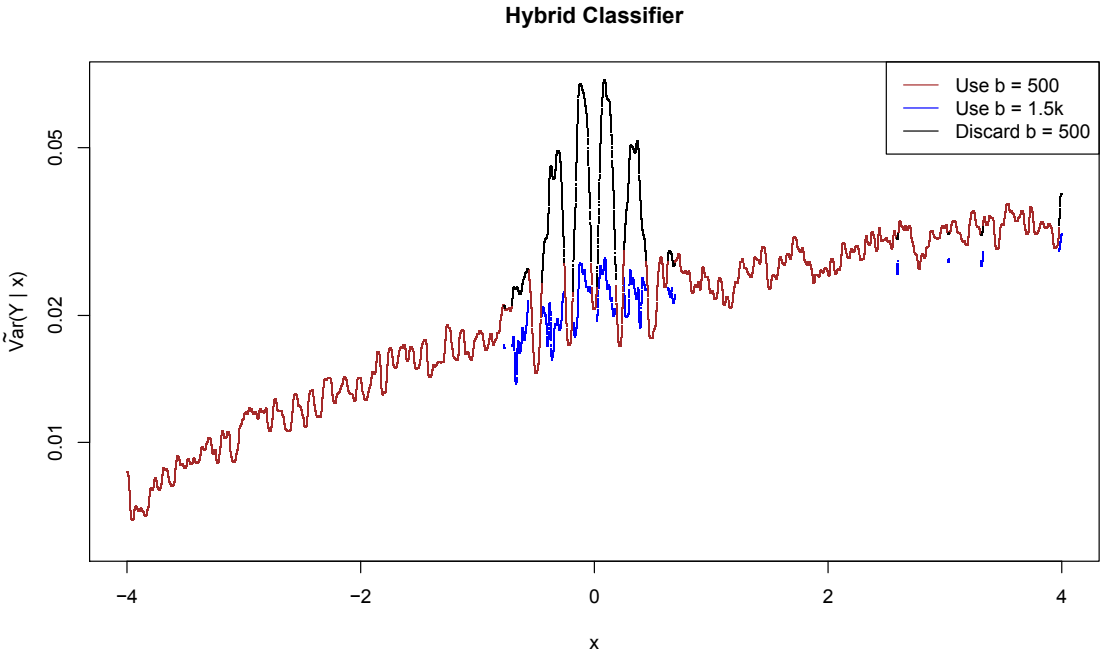


Figure 6.5 Visualization of regression classifier selection as a function of x according to thresholded estimated $Var(Y|X \in \mathcal{W}(x, b))$.

6.4 Next Steps

We proposed a model of Random Forest regression and then used it to construct a hybrid Random Forest classifier which outperforms each of several Random Forests trained using different constant bootstrap sample sizes. We must yet extend the model to higher dimensions, and a more general and more elegant formalism would be desirable. Our classifier was designed strictly for illustratative purposes, and is therefore analytically inelegant and cumbersome to produce. What would a better classifier look like which benefitted from the same model? Here is one thing it *wouldn't* look like. One might be tempted to propose a mixture classifier, using an α coefficient to blend between two different bandwidths. But we've shown that for any given choice of x, it's likely that one choice of b – call it $b^*(x)$ – will dominate all others in MSE so that the only sensible choice of $\alpha(x, b^*)$ would be 1, and every other choice of b for that x would get $\alpha = 0$. Instead we would like to learn $b^*(x)$ in a computationally efficient way. One course to pursue would be through h; learn $h(x, b)$ and try to leverage the literature on adaptive kernels to select $h(x)$, then invert $h(x, b)$. Another course would be to ponder the following observation: we hypothesize that pruning a Random Forest regression tree would be equivalent to decreasing the value of b.

Chapter 7

Conclusion

We had a finite set of actionable goals we set as we embarked on each step of this work:

- To establish a feasible methodology for producing photometric redshift estimates with confidence intervals on the errors. In Chapter 2 we demonstrated RMS error of 0.023 on the SDSS main galaxy sample, consistent with other empirical methods applied to this data. In addition, this methodology yielded zero-mean error and per-object error distribution variances which we used to produce standard normal errors – the confidence intervals we sought.

- To improve on the then-current state-of-the-art estimation errors produced by fully empirical approaches to photometric redshift estimation. In Chapter 3 we describe a reduction of the above RMS error from 0.023 to 0.017 by adding galaxy inclination as a training feature.

- To lower the training time and raise the ceiling on the size of data sets usable with Random Forest regression while holding system memory constant. In Chapter 4 we demonstrate a memory-efficient implementation, and what appears to be an asymptotically faster algorithm for training trees.

- To integrate Random Forest regression with Microsoft SQL Server. We describe a system for performing this regression in Chapter 5; due to time constraints and a shift of focus in favor of cluster computing and away from computation on monolithic machines, we chose not to follow through with the training portion of the SQL Server integration.

- To implement Random Forest regression for use on GPUs, with the particular goal of deploying on a low-power cluster. We describe such an implementation in Chapter 5.

- To demonstrate and test a hypothesis on how we might improve on the MSE performance of Random Forest regression in general. In Chapter 6 we demonstrated that bootstrap sample size has an effect analogous to bandwidth in kernel regression, and a crude methodology for automating selection between such bandwidths. On a synthetic data set designed to demonstrate this behavior this methodology successfully reduced the aggregate MSE over the single best bandwidth tested.

For future work, it is tempting yet to pursue a cluster implementation of our faster Random Forest regression tree training procedure, and indeed this seems likely. More interesting, however, would be further investigation into the relationship between Random Forest regression and kernel regression. We demonstrated and tested this relation on a particular data set designed to help understand it, but it would be satisfying and of high utility to generalize the theory and formalize the mathematical model more elegantly. Yet another seemingly ambitious, but also seemingly perfectly attainable goal for future work would be to extend the regression technique to that of the estimation of distributions of the response variable conditioned on features. What are the "kernels" we describe, but quasi-histogram estimates of precisely these distributions?

There was, in fact, one other motivation for all this which we have not yet stated: to identify and prove the conditions necessary for asymptotic consistency of Random Forest regression. This goal lies still further along the road ahead.

Bibliography

[1] K. Abazajian et al. The seventh data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 182(2):543, 2009.

[2] J. Adelman-McCarthy et al. The fifth data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 172(2):634, 2007.

[3] J. Adelman-McCarthy et al. The sixth data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, 175(2):297, 2008.

[4] H. Aihara et al. The eighth data release of the sloan digital sky survey: First data from sdss-iii. *The Astrophysical Journal Supplement Series*, 193(2):29, 2011.

[5] J. Bailin and W. E. Harris. Inclination-independent galaxy classification. *The Astrophysical Journal*, 681(1):225, 2008.

[6] W. Baum. Photoelectric magnitudes and red-shifts. In McVittie, editor, *Problems of Extra-Galactic Research, Proceedings from IAU Symposium no. 15*, page 390. Macmillan Press, 1962.

[7] N. Benítez. Bayesian photometric redshift estimation. *The Astrophysical Journal*, 536(2):571, 2000.

[8] A. Boselli and G. Gavazzi. Multifrequency windows on spiral galaxies. *Astronomy and Astrophysics*, 283:12, 1994.

[9] L. Breiman. Bagging predictors. *Machine Learning*, 24:123, 1996.

[10] L. Breiman. Random forests. *Machine Learning*, 45(1):5, 2001.

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, CA, 1984.

[12] R. Brunner, A. Connolly, and A. Szalay. The statistical approach to quantifying galaxy evolution. *The Astrophysical Journal*, 516(2):563, 1999.

[13] G. Bruzual and S. Charlot. Stellar population synthesis at the resolution of 2003. *Monthly Notices of the Royal Astronomical Society*, 344(4):1000, 2003.

[14] T. Budavári. A unified framework for photometric redshifts. *The Astrophysical Journal*, 695(1):747, 2009.

[15] T. Budavári, A. Szalay, A. Connolly, I. Csabai, and M. Dickinson. Creating spectral templates from multicolor redshift surveys. *The Astronomical Journal*, 120(3):1588–1598, 2000.

[16] S. Carliles, T. Budavári, C. Priebe, and A. Szalay. Random forests for photometric redshifts. *The Astrophysical Journal*, 712(1):511–515, 2010.

[17] G. Coleman, C.-C. Wu., and D. Weedman. Colors and magnitudes predicted for high redshift galaxies. *The Astrophysical Journal Supplement Series*, 43:393, 1980.

[18] A. Collister and O. Lahav. Annz: Estimating photometric redshifts using artificial neural networks. *Publications of the Astronomical Society of the Pacific*, 116(818):345, 2004.

[19] A. Connolly, I. Csabai, A. Szalay, D. Koo, R. Kron, and J. Munn. Slicing through multicolor space: Galaxy redshifts from broadband photometry. *The Astronomical Journal*, 110(6):2655, 1995.

[20] A. Connolly, A. Szalay, M. Bershady, A. Kinney, and D. Calzetti. Spectral classification of galaxies: an orthogonal approach. *The Astronomical Journal*, 110(3):1071, 1995.

[21] C. Conroy, D. Schiminovich, and M. Blanton. Dust attenuation in disk-dominated galaxies: Evidence for the 2175 å dust feature. *The Astrophysical Journal*, 718(1):184, 2010.

[22] I. Csabai. *Photometric Redshifts*. URL `http://www.sdss3.org/dr8/algorithms/photo-z.php`.

[23] I. Csabai, A. Connolly, A. Szalay, and T. Budavári. Reconstructing galaxy

spectral energy distributions from broadband photometry. *The Astronomical Journal*, 119(1):69, 2000.

[24] I. Csabai et al. The application of photometric redshifts to the sdss early data release. *The Astronomical Journal*, 125(2):580, 2003.

[25] H. David and H. Nagaraja. *Order Statistics*. Wiley, New Jersey, 3 edition, 2003.

[26] J. Davies and D. Burstein, editors. *The Opacity of Spiral Disks, Proceedings of the NATO Advanced Study Institute*, volume 469, Dordrecht, 1995. Kluwer Academic Publishers.

[27] J. Davies, S. Phillips, P. Boyce, and M. Disney. Selection effects or high opacity? understanding the surface brightness distribution of inclined disc galaxies. *Monthly Notices of the Royal Astronomical Society*, 260:491, 1993.

[28] L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, New York, 1996.

[29] M. Disney, J. Davies, and S. Phillipps. Are galaxy discs optically thick? *Monthly Notices of the Royal Astronomical Society*, 239:939, 1989.

[30] S. Driver, C. Popescu, R. Tuffs, J. Liske, A. Graham, P. Allen, and R. de Propris. The millennium galaxy catalogue: the B-band attenuation of bulge and disc light and the implied cosmic dust and stellar mass densities. *Monthly Notices of the Royal Astronomical Society*, 379:1022, 2007.

BIBLIOGRAPHY

[31] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1994.

[32] A. Fernández-Soto, K. Lanzetta, and A. Yahil. A new catalog of photometric redshifts in the hubble deep field. *The Astrophysical Journal*, 513(1):34, 1999.

[33] R. Giovanelli, M. Haynes, J. Salzer, G. Wegner, L. da Costa, and W. Freudling. Extinction in sc galaxies. *The Astronomical Journal*, 107(6):2036, 1994.

[34] S. Gwyn and F. Hartwick. The redshift distribution and luminosity functions of galaxies in the hubble deep field. *The Astrophysical Journal Letters*, 468(2):L77, 1996.

[35] J. Hamilton. Cooperative expendable micro-slice servers (cems): Low cost, low power servers for internet-scale services. In *Proceedings of CIDR 09*, 2009.

[36] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2001.

[37] Hildebrandt et al. Phat: Photo-z accuracy testing. *Astronomy & Astrophysics*, 523, 2010.

[38] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832844, 1998.

[39] E. Holmberg. A photographic photometry of extragalactic nebulae. *Lund Medd. Astron. Obs. Ser. II*, 136:1, 1958.

[40] J. Huizinga and T. van Albada. Extinction in sc galaxies: An analysis of the eso-lv data. *Monthly Notices of the Royal Astronomical Society*, 254:677, 1992.

[41] D. Koo. Optical multicolors: A poor person's z machine for galaxies. *The Astronomical Journal*, 90(3):418, 1985.

[42] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[43] D. Madgwick, R. Somerville, O. Lahav, and R. Ellis. Galaxy spectral parametrization in the 2df galaxy redshift survey as a diagnostic of star formation history. *Monthly Notices of the Royal Astronomical Society*, 343:871, 2003.

[44] A. Maller, A. Berlind, M. Blanton, and D. Hogg. The intrinsic properties of sdss galaxies. *The Astrophysical Journal*, 691(1):394, 2009.

[45] C. Marinoni and A. Buzzi. A geometric measure of dark energy with pairs of galaxies. *Nature*, 468(7323):539, 2010.

[46] E. Nadaraya. On estimating regression. *Theory of Probability and its Applications*, 9(1):141–142, 1964.

[47] H. Oyaizu, M. Lima, C. Cunha, H. Lin, and J. Frieman. Photometric redshift error estimators. *The Astrophysical Journal*, .

[48] H. Oyaizu, M. Lima, C. Cunha, H. Lin, J. Frieman, and E. S. Sheldon. A galaxy photometric redshift catalog for the sloan digital sky survey data release 6. *The Astrophysical Journal*, .

[49] N. Padilla and M. Strauss. The shapes of galaxies in the sloan digital sky survey. *Monthly Notices of the Royal Astronomical Society*, 388:1321, 2008.

[50] N. Padmanabhan et al. An improved photometric calibration of the sloan digital sky survey imaging data. *The Astrophysical Journal*, 674(2):1217, 2008.

[51] B. Ryden. The ellipticity of the disks of spiral galaxies. *The Astrophysical Journal*, 601(1):214, 2004.

[52] M. Sawicki, H. Lin, and H. Yee. Evolution of the galaxy population based on photometric redshifts in the hubble deep field. *The Astronomical Journal*, 113 (1):1, 1997.

[53] Z. Shao et al. Inclination-dependent luminosity function of spiral galaxies in the sloan digital sky survey: Implications for dust extinction. *The Astrophysical Journal*, 659(2):1159, 2007.

[54] C. Stoughton et al. Sloan digital sky survey: Early data release. *The Astronomical Journal*, 123(1):485, 2002.

[55] M. A. Strauss et al. Spectroscopic target selection in the sloan digital sky survey: The main galaxy sample. *The Astronomical Journal*, 124(3):1810, 2002.

[56] A. Szalay, G. Bell, et al. Graywulf: Scalable clustered architecture for data intensive computing. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, 2009.

[57] A. Szalay et al. Low power amdahl-balanced blades for data-intensive computing. *ACM SIGOPS Operating Systems Review*, 44(1):71, 2010.

[58] C. Unterborn and B. Ryden. Inclination-dependent extinction effects in disk galaxies in the sloan digital sky survey. *The Astrophysical Journal*, 687(2):976, 2008.

[59] E. Valentijn. Opaque spiral galaxies. *Nature*, 346(6280):153, 1990.

[60] Y. Wang, N. Bahcall, and E. Turner. A catalog of color-based redshift estimates for $z \lesssim 4$ galaxies in the hubble deepfield. *The Astronomical Journal*, 116(5): 2081, 1998.

[61] L. Wasserman. *All of Nonparametric Statistics*. Springer, New York, 2006.

[62] G. Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, 26(4):359–372, 1964.

[63] C. Yip, A. Szalay, R. Wyse, L. Dobos, T. Budavári, and I. Csabai. Extinction in star-forming disk galaxies from inclination-dependent composite spectra. *The Astrophysical Journal*, 709(2):780, 2010.

[64] C. Yip et al. Spectral classification of quasars in the sloan digital sky survey: Eigenspectra, redshift, and luminosity effects. *The Astronomical Journal*, 128 (6):2603, 2004.

[65] C.-W. Yip et al. Effect of inclination of galaxies on photometric redshift. *The Astrophysical Journal*, 730(1):54, 2011.

[66] D. York et al. The sloan digital sky survey: Technical summary. *The Astronomical Journal*, 120(3):1579, 2000.

Vita

Samuel Carliles recieved his B.S. degree in Computer Science from the Johns Hopkins University in 2003, and enrolled in the Computer Science Ph.D. program at the Johns Hopkins University in 2005. In 2013 he began work as a Data Scientist at Mediaglu, LLC, which was acquired by AppNexus, Inc. in 2015. His research focuses on statistical regression methodology and scalable implementation.